The PAN book

Federico Bizzarri
Angelo Brambilla
Giancarlo Storti Gajani

version 2014.3.a

Copyright © 2014 Giancarlo Storti Gajani

PUBLISHED BY PUBLISHER

BOOK-WEBSITE.COM

Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at `http://creativecommons.org/licenses/by-nc/3.0`. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, March 2015

Contents

1 Preface 7

2 A simple PAN walk through 9

2.1 Overview **9**
2.1.1 Where and How: downloading and installing 9
2.1.2 On line help and quick reference 9
2.1.3 Netlist Files 10
2.1.4 Models and devices 11
2.1.5 Global Options 11
2.1.6 Output format 11

2.2 The DC analysis **12**
2.2.1 A very simple example 12
2.2.2 DC sweep and device models 16
2.2.3 The `alter` pseudo-analysis 17
2.2.4 Multiple DC sweeps 19
2.2.5 Multiple DC operating points (and simulator directives) 20

2.3 The time domain analysis `tran` **23**
2.3.1 A Van der Pol oscillator, parameters, expressions, and the `poly` device 23
2.3.2 A simple linear LC oscillator and frequency warping effects 26

2.4 Envelope time domain analysis: envelope **27**
2.4.1 The Bubba oscillator, using Schematic Capture and Netlisting tools 27

2.5 Steady state time domain analysis: shooting **29**
2.5.1 The Butler overtone oscillator 30

3 AMS simulation with PAN 35

3.1 A simple Analog Digital oscillator **35**

3.2 The `pnoise` analysis **38**

4 Reference 41

4.1 Where and How: downloading and installing 41

4.2 Basic one-port device 41
4.2.1 Linear Two Terminal Resistor 41
4.2.2 Linear Two Terminal Capacitor 43
4.2.3 Two Terminal Inductor 44
4.2.4 Independent Voltage Source 45
4.2.5 Independent Current Source 47

4.3 Basic two-port device 49
4.3.1 Voltage Controlled Current Source 49
4.3.2 Linear Voltage Controlled Voltage Source 50
4.3.3 Linear Current Controlled Current Source 51
4.3.4 Current Controlled Voltage Source 52
4.3.5 Linear Two Winding Ideal Transformer 54
4.3.6 Ideal Nullor 55
4.3.7 Coupled Inductors 55
4.3.8 Polynomial Transconductor 55

4.4 semiconductor device 56
4.4.1 Junction Diode 56
4.4.2 Bipolar Junction Transistor 58
4.4.3 Junction Field Effect Transistor 60
4.4.4 Philips M903 mosfet transistor 61
4.4.5 MOSFET level 2 device 63
4.4.6 MOSFET level 3 device 65
4.4.7 Berkeley Short Channel IGFET Model Version-3 66
4.4.8 Berkeley Short Channel IGFET Model Version-4.50 76
4.4.9 GaAs MESFET 93
4.4.10 Philips juncap diode model 94

4.5 misc device 95
4.5.1 analog to digital converter 95
4.5.2 analog to digital converter 95
4.5.3 Current probe 96
4.5.4 Microstrip Line 97
4.5.5 Linear N Port 97
4.5.6 Independent Resistive Source 99
4.5.7 Poly Silicon Thin Film Transistor 100
4.5.8 Time controlled switch 102
4.5.9 Transmission Line 103
4.5.10 Voltage controlled switch 104

4.6 Basic analyses 105
4.6.1 DC Analysis 105
4.6.2 Small Signal Analysis 109
4.6.3 Time domain analysis 109

4.7 Simulator control pseudo-analyses 115
4.7.1 Set Pan Options 115
4.7.2 Alter an instance parameter or simulation option 117
4.7.3 Sweep Analysis 117
4.7.4 Control block 118

4.7.5 Monte Carlo Analysis . . . 122

4.8 Advanced time domain analyses 123

4.8.1 Time domain analysis with envelope algorithm . . . 123
4.8.2 Steady state computation through shooting method . . . 126
4.8.3 Steady state computation through the multi-tone shooting method . . . 132
4.8.4 Periodic Small Signal Analysis . . . 134
4.8.5 Noise analysis of a periodic circuit . . . 135

4.9 Advanced frequency domain analyses 137

4.9.1 Pole Zero Analysis . . . 137
4.9.2 Periodic Steady-State Analysis . . . 137
4.9.3 Almost-Periodic Steady State Analysis . . . 140
4.9.4 Small Signal S-Parameter Analysis . . . 142
4.9.5 Noise analysis of a periodic circuit . . . 143

biblio . . . 145

Books 145

Articles 145

Technical Reports 146

Index . . . 147

1. Preface

This book represents a “hands on” introduction to the PAN circuit simulator. Since our goal was to provide a fast track to potential users it has been formatted as a “walkthrough” that starts from the simplest concepts and evolves towards more complex methods and analyses. PAN is an accademic simulator, it has many powerful features, it is free, but, obviously, most certainly has bugs of which we are not aware. We are quite eager to correct them, and also to implement new user requested features, but users may not expect the same kind of (excellent) support that comes with commercial simulators.
PAN has also a few features that, to our knowledge, are not implemented in currently available commercial simulators. The most notable of these is the capability to perform steady state and noise analyses on circuits that are built using a mixture of analog, digital and behavioural blocks. A simple example of the use of these algorithms is shown in a separate chapter.

IMPORTANT NOTE: this book is work in progress !

In its current status it can be used to start using PAN in a simple and direct way, we intend to expand it with many details and considerations on the algorithms that, currently, are missing. If you are interested please check on the PAN distribution site for new versions.

2. A simple PAN walk through

2.1 Overview

The PAN simulator is mainly a circuit simulator but can be easily extended and controlled in order to perform several different types of system simulations.
PAN is capable of analog, digital or AMS simulation, electrothermal simulation and can be extended by linking external modules.
A complete list of anayses, devices and directives that are available natively (i.e. without *user extensions*) are listed in the reference section at the end.

This chapter represents a sort of tutorial showing in the most direct way and through a limited number of examples, how to use the PAN circuit simulator to perform simple simulation tasks. More complex tasks and a detailed definition of all parameters proper to each analysis method will be given in later chapters.

2.1.1 Where and How: downloading and installing

The PAN circuit simulator binary can be downloaded from `http://brambilla.ws.dei.polimi.it`.
The PAN binary requires a standard linux installation. PAN developers use Debian based systems (most recently Debian, Mint and Ubuntu), but PAN has been used without any pain with other Linux flavours, including RedHat, Gentoo, Sabayon and several others.

A Linux live CD image including the latest version of the simulator and of the demo files is also available. This allows quick evaluation without actually installing anything in your computer.

Using PAN is relatively simple, and only requires knowledge of basic Linux terminal commands and the use of any ASCII editor (such as vi). As it is netlists are entered in ASCII format, but, as it will be shown in the sequel, thanks to the high compatibility of PAN with well known simulator languages, almost any standard netlister can be used, as long as some adjustments to the command language are made.

2.1.2 On line help and quick reference

A quick reference is available with

```
prompt> pan -s
```

a list of keywords appears, and further help on a specific topic can be obtained with the command

```
prompt> pan -s topic
```

Finally, command

```
prompt> pan -h
```

outputs to terminal a brief list of command line options.
Most problems a user may encounter can be solved using the on line help.

2.1.3 Netlist Files

PAN automatically understands different netlist formats, this means that the circuit netlist can be written using standard schematic capture software. PAN specific commands, analysis and options must be added manually. In the following examples, PAN "native" format will be used, it is very simple and (almost) free format.
PAN netlist and control files are usually written using extension *.pan*, and simulation execution is, in the most simple case, started with
prompt> pan *netlist.pan*.

The few basic rules that any user needs to know in order to write simple netlist files are:

Comments Comments start from character ";" and continue till the end of line

Devices Each device instance is described by a single line, starting with the *instance name* (that must, obviously, be unique), then the list of nodes to which the instance is connected, the device type (e.g. "resistor", "capacitor" etc.), and, finally, instance specific parameters and options. More details are given in the following.

Analysis Analysis are in general written on a single line. The first token on the line must be a unique identifier, the second is the analysis type (e.g. dc, tran etc.), finally all analysis parameters and options.

Long lines For formatting or aesthetic purposes it is possible to break long lines using character "\".

Nodes Nodes are implicitly defined by the instance definition lines, the only exception is the reference node that must be explicitly defined using the netlist line (or "card" as in the old spice tradition)

```
ground electrical name
```

where *name* can be any unique identifier.

Options and Parameters device and analysis parameters and options are in general written in the form

```
name = value
```

separated by spaces. Boolean values can be entered also using keywords yes or no that correspond, respectively, to 1 and 0.

More complex rules, such as those needed to define subcircuits, models, or control structures, will be shown later. These simple rules are exemplified below in Netlist 2.1.1

Netlist 2.1.1 — Basic netlist writing rules. –

```
ground electrical 0   ; This is a comment

; Circuit
r1  n1 n2      resistor  r=1k       ; a resistor instance
InstanceName    Node1 Node2 Node3 Node4 \
```

```
    Node5 Node6 Node7  MultiportDevice option1=10 \
    option2=30  ; Instance of MultiportDevice with line breaks
```

2.1.4 Models and devices

Any circuit is essentially described by a topology and a set of components. Components can be defined in many different ways. PAN includes a large number of predefined standard devices, several customizable devices, and the possibility to define completely new devices by writing code in many different languages. Moreover, any device, predefined or user defined, can be made in to a template using the `model` statement. This is a standard feature of most spice like simulators.

Netlist 2.1.2 — An example of definition and use of a diode model. –

```
ground electrical 0

; Circuit
r1  n1  0      resistor  r=1k
d1  n1  0      MYDIODE
d2  n1  0      MYDIODE bv=100 imax=100

; Models
model MYDIODE diode is=3.142f rs=4.2 bv=80
```

An example of model definition and use is shown in Netlist 2.1.2, note how diode instance `d1` is defined by the model template "as is" while diode `d2` is based on model `MYDIODE` but has the additional `imax` parameter defined and parameter `bv` redefined (instance parameters override those defined in the model).
Definition of new components will be discussed in a later section.

2.1.5 Global Options

Global simulation options, such as temperature, convergence aid parameter values (gmin, gmax, gground...), topology checker options, global verbosity level and many more are defined with line:

```
name options option1=value1 option2=value2 ...
```

where *name* is a optional unique identifier and as many space separated options as desired can be given after the `options` keyword.

2.1.6 Output format

In general PAN will output to terminal and to a log file a considerable amount of messages. These include error messages, information on the simulator current activity and, in some cases, some information about simulation results. Terminal output can be limited to error messages with option "-q", i.e.

```
prompt> pan -q netlist.pan
```

will result in a log file *netlist.log* and terminal output containing only few notices concerning simulator status and eventual error messages. Simulation results for each analysis in *netlist.pan* is stored in a directory named *netlist.raw*. This directory contains an ASCII

index file and one or more output file for each analysis each named with the unique analysis identifier specified in the netlist file.

These files are in standard SPICE3 rawfile format and can thus be viewed using several visualization tools such as the open source tools gwave, KJwave, or commercial tools such as Synopsys sx or CustomWave. The visualization tool iplot is also available at the PAN dowload site. The tool is based on a rather old, but still quite efficient, set of libraries.

It is also possible to import rawfiles into Matlab or Octave using one of the several rawfile import filters easily available on the web.

2.2 The DC analysis

The DC analysis computes the operating point of linear and nonlinear circuits. The default behaviour of the simulator is to compute one single operating point, even if the circuit admits more than one, but analysis also implements methods to look for more than one operating point. Like most methods implemented in PAN it is based on the method known as Modified Nodal Analysis (MNA) [HRB75], this means that the default behaviour of the simulator is to compute only node voltages and current of circuit ports that are not voltage controlled, these are often called "bad branches". An introduction to MNA can be found in many classic books, see e.g. [CDK87], [VS83].

Obviously, if we wish to explicitly compute the value of a specific current of the circuit, the simplest solution is to introduce a null independent voltage source in series with branch whose current we are interested in.

2.2.1 A very simple example

As a first example, consider the circuit in Figure 2.1 and the corresponding Netlist 2.2.1

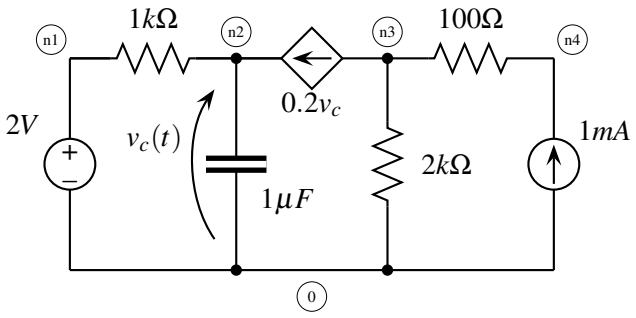

Figure 2.1: A simple example.

Netlist 2.2.1 — Simple netlist. – File: Simple_001.pan –

```
ground electrical 0

; Analyses
Dc0 dc print=yes

; Circuit
r1  n1 n2      resistor  r=1k
c1  n2  0      capacitor c=1u
r2  n3  0      resistor  r=2k
```

```
r3  n3 n4      resistor  r=100
tc1 n3 n2 n2 0 vccs      gain1=0.2
v1  n1  0      vsource   vdc=2
i1  n4  0      isource   idc=1m
```

Each device instance in this circuit is entered in its own line and correspondence of figure to netlist should be self evident. The two-port voltage controlled current source is defined by first giving, respectively, the sink and source pins, followed by the sense nodes, each pair of sense nodes requires a corresponding `gain` parameter. Controlled sources can be described also in other ways, please refer to the output of command:

```
prompt> pan -s vccs
```

(or the same command with `vcvs`, `ccvs` or `cccs`) for more information.
The DC analysis command is on a single line:

```
Dc0 dc print=yes
```

The `Dc0` is a unique identifier, more than one DC analysis with different options can be issued in the same file, each must have its unique name. Option `print=yes` instructs the simulator to print the DC solution that has been found to output. Default value of this parameter is `no` and, in this case, the operating point is written only in the output file. Direct terminal output can be useful, but, in general, is practical only for small circuits.
Let us now execute the simulation:

```
prompt> pan Simple_001.pan
```

the output of this command consists in terminal output, a log file (that is essentially equivalent to the screen output) and an output directory named `./Simple_001.raw` that contains results of all simulations in the `PAN` file, plus an index file.
In general log files are used only for debugging purposes, since this is the first example that we are going to see let's look at the output with some detail.

Logfile 2.2.2 — Logfile of Simple netlist. – Simple_001.log –

```
Reading circuit from file <Simple_001.pan> ...
Netlist successfully read ...
Building global model declarations ...
Building instance and node declarations ...
Mapping node names ...
Building analysis and option declarations ...
Checking for undefined nets ...

NODE and BRANCH SUMMARY:
      4 Nodes

      5 Total equations

DEVICE SUMMARY:
```

```
      1 capacitor
      1 isource
      3 resistor
      1 vccs
      1 vsource

Reference node is <0>.

**********************
* Dc analysis: <Dc0>. *
**********************
Checking circuit topology for DC analysis...
Installing <gground> resistors ...

Finding the initial guess for the Newton algorithm ...
Start of DC analysis ...
..

Total power dissipated by independent sources is <2.1000 mW>.

Convergence achieved in <2> iterations.
Current solution of <Dc0> DC analysis:

  V(n1)           =  2.0000  V
  V(n2)           = -10.050 mV
  V(n3)           =  2.0201  V
  V(n4)           =  1.9201  V
  I(v1:i)         = -2.0101 mA

CPU time for this analysis was 0.00 seconds.
Elapsed time was 0.00.
Physical memory usage was 296 kBytes.

Resource usage:
    CPU user time:        0.00 seconds.
    CPU system time:      0.00 seconds.


Execution date: Sat May 31 15:57:54 2014
```

PAN reads the netlist and, after a number of messages asserting execution status, prints a circuit structure summary. In our case 4 nodes and 5 equations (node count does not consider the reference node). The equations are 5 since the vsource is a “bad branch”, i.e. not voltage controlled, and thus requires the introduction of an additional equation. The device summary section reports the number of device instances found and is quite interesting when large circuits are simulated.

After these preliminary steps analysis are performed one at a time in the order they are encountered. In our case we have one single DC analysis called Dc0. After a few messages (about gground resistors, and initial guess for the Newton algorithm, the meaning of these

messages will be discussed later) DC simulations is performed. Total power absorbed by independent sources is always output to log and screen, current solution, i.e. the 4 node voltages and the current in the voltage source, is output only if parameter `print` is set to true. Finally some execution time statistics are shown.

The `annotate` option

All analysis admit the option `annotate` that controls the amount of output generated. This parameter is strongly context dependent and allows the user, in many cases, to output information not in general available in most commercial simulators. The general format of the `annotate` option is a decimal value from 0 to 7 that correspond to increasing level of verbosity, plus several bitwise values that act as flags and can be added. These flags can be entered as hex constants and are used to enable output of context dependent information. As an example, modify the 4th row of `Simple_001.pan` so that it reads

```
Dc0 dc print=yes annotate=3+0x40+0x80
```

in this case the standard verbosity level "3" is chosen but, in addition, (value `0x40`) the simulator is instructed to print the MNA matrix to standard output and to the log file at each Newton iteration step; finally (value `0x80`) the MNA matrix is dumped to file `Dc0.m`. A large amount of information is printed, in our case, the last lines of the log file are:

Logfile 2.2.3 — Logfile. – `Simple_001.log` (with `annotate=3+0x40+0x80`) –

```
....
-------- matrix dump at iteration <2> --------
MATRIX SUMMARY

Matrix is 5 x 5 and real.

Matrix before decomposition:
              1          2          3          4          5

   1      0.001     -0.001        ...        ...          1
   2     -0.001       -0.2     -1e-12        ...        ...
   3        ...        0.2      0.011      -0.01        ...
   4        ...        ...      -0.01       0.01        ...
   5          1        ...        ...        ...        ...


Largest element in matrix = 1.
Smallest element in matrix = 1e-12.

Largest pivot element = 1.
Smallest pivot element = 0.01.

Number of elements = 12.

Density = 48.00%.
Number of fill-ins = 0.
```

```
-------- RHS vector --------
Beginning source vector.
1    4.25689263434929e-19
2    3.02872507961481e-19
3    1.71973044932689e-19
4    8.58284610952035e-20
5    0


-------- Delta solution vector --------
Beginning source vector.
1    0
2    -1.5219724074667e-18
3    1.1243919682759e-15
4    1.13297481427212e-15
5    4.24167291027462e-19

Current solution of <Dc0> DC analysis:

  V(n1)           =  2.0000  V
  V(n2)           = -10.050 mV
  V(n3)           =  2.0201  V
  V(n4)           =  1.9201  V
  I(v1:i)         = -2.0101 mA

    Matrix element number:       12
    Fill-ins number:              0
    Sparsity percentage:         52
    Reordering number:            1
    Exchaged rows/cols:           5
    Subcircuit solutions:         0

....
```

2.2.2 DC sweep and device models

A common use of the DC analysis is not limited to the computation of one single operating point, but to the computation of operating points as the value of one circuit parameter is varied. This analysis mode is known as *sweep* and most simulators provide a DC sweep analysis. In PAN DC sweep can be performed through a number of different options that control the parameter or option to be swept, and the sampling of the values to be used for the sweep (i.e. linear, logarithmic ...). Instance parameters (i.e. resistance value of a resistor, value of an independent source...) or simulator options (e.g. temperature) may be swept.

Sweep, among other applications, can be used to plot the DC characteristic of a non linear component. In this case we wish to plot the characteristic of an instance of a diode described by some of its process parameters. We thus add the model line

```
    model mydiode diode is=3.142f rs=3.536 n=1 xti=3 eg=1.11 \
              m=0.4186 vj=0.75 fc=0.5 bv=80 ibv=24.573m imax=100
```

where the model of a device of type `diode` and named `mydiode` is created. A quick reference to

the meaning of diode parameters in the line above is given by command `pan -s diode`. All these parameters are compatible with those found in standard commercial component libraries. Now `mydiode` can be used as any other diode component. More than one diode model can be specified in each netlist file.

Netlist 2.2.4 — Simple netlist. – File: `Simple_002.pan` –

```
ground electrical 0

; Analyses
Dc0 dc print=yes instance=v1 param="vdc" start=-85.0 \
        step=0.01 stop=10.0

; Circuit
v1  n1  0      vsource   vdc=2
d1  n1  0      mydiode

; Models
model mydiode diode is=3.142f rs=3.536 n=1 xti=3 eg=1.11 \
m=0.4186 vj=0.75 fc=0.5 bv=80 ibv=24.573m imax=100
```

The `Dc0` analysis line instructs the simulator to perform a (large) number of independent DC analysis, sweeping parameter "`vdc`" of instance `v1` from -85.0V to 10.0V with steps of 100mV. The resulting output (imported and plotted using Matlab) is shown in Fig. 2.2.

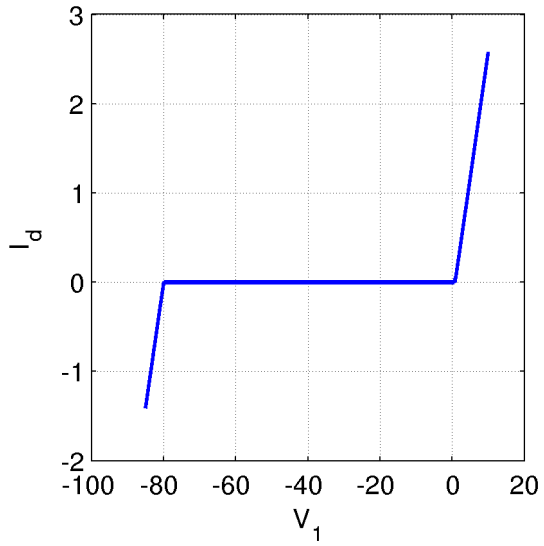

Figure 2.2: Diode current vs. generator voltage, generated by Netlist 2.2.4.

2.2.3 The `alter` pseudo-analysis

Even if `alter` is listed among the analysis and follows the same syntax, it is actually a directive, instructing the simulator to modify a parameter or option value. The typical use of this statement is that of performing the same analysis with different values of circuit or device parameters or

global options[1]. Consider once more the diode DC sweep example, but assume we are interested in comparing its characteristic at different temperatures. A simple way to do this is shown in Netlist 2.2.5.

Netlist 2.2.5 — Diode DC sweep with alter statements. – File: Simple_003.pan –

```
ground electrical 0

; Analyses
Dc0 dc print=yes instance=v1 param="vdc" start=-85.0 \
        step=0.01 stop=10.0

Alter1 alter opt=yes param="temp" value=-10

Dc1 dc print=yes instance=v1 param="vdc" start=-85.0 \
        step=0.01 stop=10.0

Alter2 alter opt=yes param="temp" value=120

Dc2 dc print=yes instance=v1 param="vdc" start=-85.0 \
        step=0.01 stop=10.0

; Circuit
v1  n1  0      vsource   vdc=2
d1  n1  0      mydiode

; Models
model mydiode diode is=3.142f rs=3.536 n=1 xti=3 eg=1.11 \
m=0.4186 vj=0.75 fc=0.5 bv=80 ibv=24.573m imax=100
```

Three subsequent DC analysis are performed at three different temperatures: the first at nominal temperature (i.e. 27 Celsius), the second at $T = -10$ and the third at $T = 120$. Results, zoomed in to show the right knee area, are shown in Fig. 2.3. The `alter` statement in netlist 2.2.5 modifies the value of a global simulation option:

```
    Alter1 alter opt=yes param="temp" value=-10
```

device parameters are modified using a similar syntax; to modify resistor instance `r1` a possible `alter` card is shown below:

```
    ⋮

    r1  n1  n2  resistor    r=1k

    ⋮

    Alter_r1 alter instance=r1 param="r" value=10k
```

[1] Recall that a list of options and parameters can be quickly obtained using command `pan -s` *name* where *name* is the keyword `options` for global simulator options or the name of the device, or of the analysis for specific options.

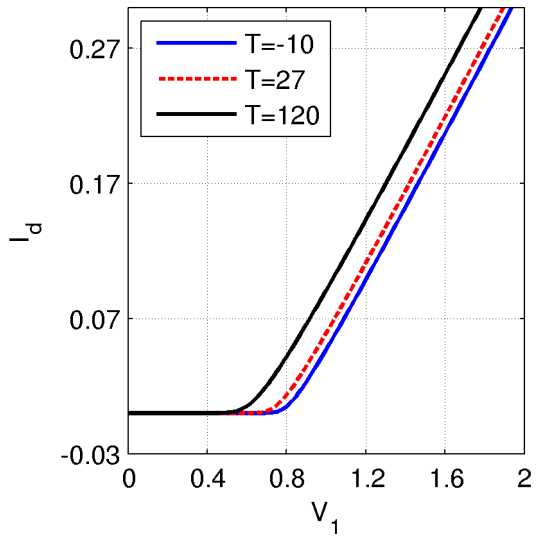

Figure 2.3: Diode current vs. generator voltage, at three different temperatures, generated by Netlist 2.2.5.

```
    ⋮
```

2.2.4 Multiple DC sweeps

In many cases one may desire to sweep two different circuit parameters at the same time, this is possible by simply adding more than one sweep instruction in the same analysis (any number of embedded sweeps are possible, even if, possibly impractical and difficult to analyse). A simple circuit where two parameters are swept at the same time is shown in Netlist 2.2.6.

Netlist 2.2.6 — Double DC sweep. – File: `Simple_004.pan` –

```
ground electrical 0

; Analyses
Dc0 dc print=yes \
    instance=v1 param="vdc" start=-85.0 step=0.01 stop=10.0 \
    instance=r1 param="r" start=1.0 step=2.0 stop=11.0

; Circuit
v1  n1  0      vsource   vdc=2
d1  n1  0      mydiode
r1  n1  0      resistor  r=1

; Models
model mydiode diode is=3.142f rs=3.536 n=1 xti=3 eg=1.11 \
m=0.4186 vj=0.75 fc=0.5 bv=80 ibv=24.573m imax=100
```

The resulting output is stacked columnwise in output file `Simple_004.raw/Dc0`, after destacking (most wave viewer software can take care of this) we have a set of waveforms such as those shown in Fig. 2.4.

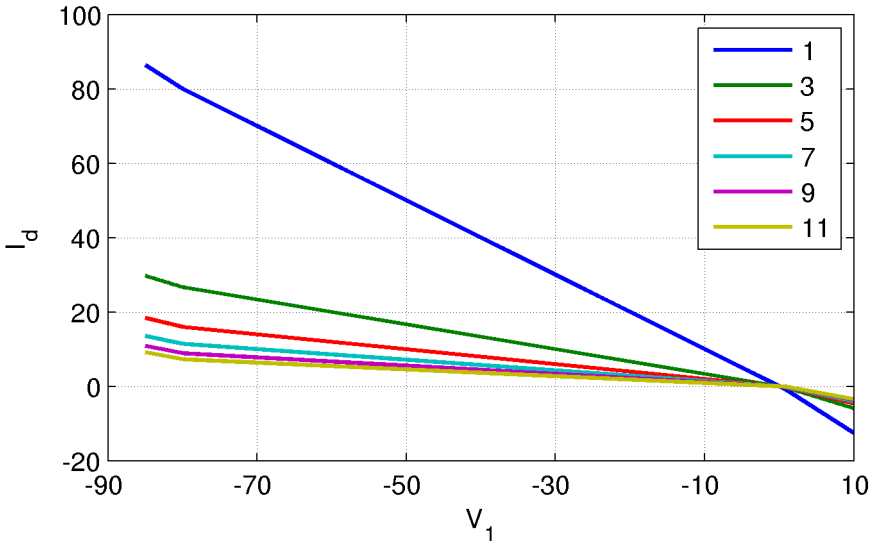

Figure 2.4: Voltage source current vs. generator voltage for different values of parallel resistance (Netlist 2.2.6).

2.2.5 Multiple DC operating points (and simulator directives)

We now present a non standard option of PAN 's DC analysis algorithm. Even if this option may seem somewhat beyond the scope of a walk-through, we believe it is sufficiently simple and important to be included here.

Most non-linear circuits can have more than one DC solution. These solutions correspond to operating points, and, in time domain, to equilibria. PAN implements a heuristic method that can find multiple operating points when they are present. Note that the algorithm may not find all solutions, in some cases, but has proven to work quite nicely in many situations

To our knowledge, in current commercial or non commercial simulators, search for multiple operating points is not implemented.

Consider the voltage regulator shown in Fig. 2.5. whose netlist is shown in 2.2.7.

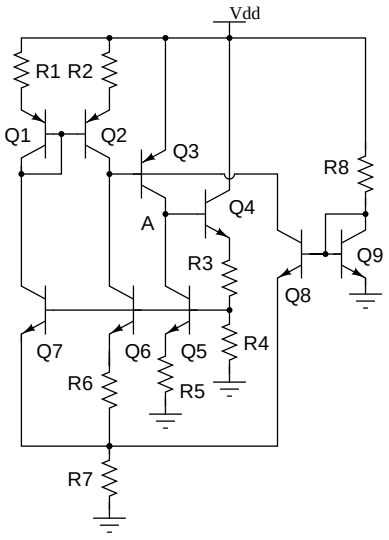

Figure 2.5: A simple voltage regulator.

Netlist 2.2.7 — Multiple DC solution. – File: `vregu.pan` –

```
// Voltage regulator.
ground electrical gnd

; Analyses
; Find possible multiple solutions
Dc1 dc print=yes multiple=yes stab=1

; Circuit
vcc    vcc    gnd   vsource vdc=20
r1     vcc      a   resistor r=5k
q1       b  b   a   vcc P
r2     vcc      l   resistor r=5k
q2       c  b   l   vcc P
q6       b  d   e   gnd N
q7       c  d   f   gnd N area=5
r3       f      e   resistor r=1k
r4       e    gnd   resistor r=5k
q3       g  c vcc   vcc P
q5       g  d   o   gnd N
r5       o    gnd   resistor r=5k
q4     vcc  g out   gnd N
r6     out      d   resistor r=3.54k
r7       d    gnd   resistor r=1k

#ifdef STARTUP

; Startup
q8       c  i   e   gnd P
q9       i  i gnd   gnd N
r8     vcc      i   resistor r=100K

#endif

; Models
model P bjt pnp=yes bf=200  vaf=50
model N bjt npn=yes bf=600  vaf=100
```

The circuit is a simple voltage regulator, two simple `BJT` models are used, the remaining part of the netlist is quite simple and straightforward, except for the part enclosed in the two statements:

```
#ifdef STARTUP

... devices, analyses, options...

#endif
```

`PAN` allows the use of "Simulator directives". These instructions allow conditional inclusion of device, analysis and option declarations. In our example these lines are included if symbol

STARTUP is defined, and inclusion of these lines will activate a startup circuit. Symbols can be defined inside the netlist using, in our case, with instruction

```
#define STARTUP
```

or on the command line, i.e.,

```
prompt>  pan vregu.pan -ad STARTUP
```

where command line option "-ad" stands for *analog define*. Let us now simulate this voltage regulator *without* using the startup circuit. We obtain the logfile shown (only a small in part) below.

Logfile 2.2.8 — Logfile of the voltage regulator. – vregu.log –

```
....
The computed equilibrium point is stable: pole at -1.70e+38Hz

Current solution of <Dc1> DC analysis:

  V(vcc)          =  20.000  V
  V(a)            =  20.000  V
  V(b)            =  19.668  V
  V(l)            =  20.000  V
  V(c)            =  19.549  V
  V(d)            =  3.7595 mV
  V(e)            =  196.12 nV
  V(f)            =  215.68 nV
  V(g)            =  642.26 mV
  V(o)            =  3.2307 nV
  V(out)          =  17.068 mV
  I(vcc:i)        =  0.0000  A

V[0] =  1.800  V (1  10.0 %).
Sweeping status:
    V[0] =  2.5414  V RHS = -16.713 pA | RHS | =  1.1206 uA

The computed equilibrium point is stable: pole at -1.70e+38Hz

... ...

  V(f)            =  3.5502  V
  V(g)            =  19.964  V
  V(o)            =  3.4561  V
  V(out)          =  19.152  V
  I(vcc:i)        =  0.0000  A

The number of found solutions is 5.

....
```

In the full log file five solutions are shown, and stability of each solution is computed by local linearization in the equilibria. If the same circuit is simulated with STARTUP defined, only one solution is found. We do not discuss here some of the messages displayed in the logfile,

since they can be ignored unless convergence problems arise.

2.3 The time domain analysis tran

The tran analysis computes the evolution in time of a circuit. Since it is based on MNA, it will actually compute the evolution of all node voltages and only of currents in "bad-branches".

2.3.1 A Van der Pol oscillator, parameters, expressions, and the poly device

Consider the Van der Pol oscillator shown in Fig. 2.6 whose netlist is shown in 2.3.1.

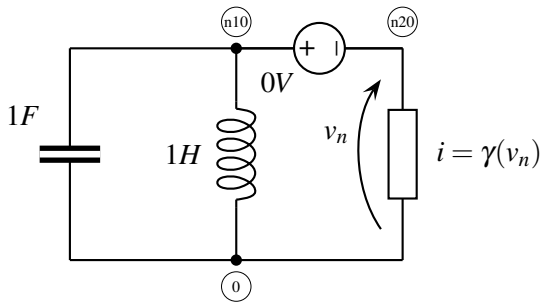

Figure 2.6: A Van der Pol oscillator

Netlist 2.3.1 — Multiple DC solution. – File: vanderpol.pan –

```
ground electrical 0

parameters R=10 ALPHA=1/3

; Analyses
Tran0 tran tstop=100 uic=1
Tran1 tran tstop=120 restart=no

; Circuit
r1    n20     0  n20   0   poly n0=0 n1=-1/R n2=0 n3=ALPHA/R d0=1
vr1   n10   n20  vsource   vdc=0
c1    n10     0  capacitor c=1 icon=3
l1    n10     0  inductor  l=1
```

Before we perform the simulation and discuss the tran analysis line, let's briefly comment some interesting features of this netlist.

Parameters and Expressions

First of all, note that all numerical values can be defined using parameters.

```
parameters R=10 ALPHA=1/3
```

It is possible to define as many parameters as desired, parameters can be defined as a mathematical expression that may contain other parameters. Any parameter can then be used in the netlist, and combined in expressions. PAN expressions can be formed using the standard four arithmetic operators plus a number of standard mathematical functions, for a full list type

```
prompt> pan -s expressions
```

The output is somewhat terse, but should allow to solve most problems one may encounter. Expressions are allowed in all fields where a number is expected.

The nonlinear transconductance device poly

PAN also allows to define rational transconductances. These are defined using device poly. It is a nonlinear voltage controlled current source two port. The first port, i.e. the first two node names, represent the current source, while the second pair the "sense" nodes. In our case, since we want to realize a simple non-linear one port, the two ports are the same. Coefficients for numerator and denominator of the rational function are given using parameters n0, n1 ... n8 and d0, d1 ... d8 respectively. Not specifying any denominator coefficient is equivalent to d0=1, on the other hand, not specifying any numerator coefficient results in anull polynomial. This means that polynomials up to degree 8 can be defined. In our case the netlist line:

```
r1 n20 0 n20 0 poly n0=0 n1=-1/R n2=0 n3=ALPHA/R d0=1
```

represents function $i = \alpha/Rv^3 - 1/Rv$.

Voltage sources as current measure devices

A null independent voltage source has also been introduced in series with the non linear element, this forces MNA to explicitly add the current in the generator as a variable providing its value in the output data file.

Initial Conditions and simulation of the Van der Pol oscillator

Let us finally consider the *first* analysis line (we will consider the second one later):

```
Tran0 tran tstop=100 uic=1
```

As always, the first word Tran0 represent a unique name for the analysis while the second is the analysis type, in our case the keyword tran, asserting that this is a *transient* aka *time domain* analysis. Option tstop is mandatory: it represents the time at which simulation must stop. In our case, since this is a rather "slow" oscillator, the rather large tstop = 100s is used.
The second option, uic=1, stands for "use mode 1 for Initial Conditionss (ICs)". Three different IC modes are allowed:

uic=0 is equivalent to "no ICs", i.e. perform a DC analysis to set ICs,

uic=1 use ICs given for all capacitors and inductors and set to 0 those that are not defined (i.e. 0 voltage on capacitors and 0 current in inductors)

uic=2 use ICs given for all capacitors and inductors and assume that all capacitors and inductors whose initial conditions have not been specified are initially considered equivalent to open and short circuits respectively.

Initial conditions are set on each capacitor and inductor line with option icon=*value*. In our case the IC has been set for the capacitor, while the inductor has been left unspecified; since mode 1 is set, this means that the initial current value in the inductor is set to 0.
After simulation (the command is pan vanderpol.pan, from now on we will specifiy the simulation command only when non-standard), we find the folder vanderpol.raw containing the usual index file and files Tran0 and Tran1 containing simulation results.

Plotting the inductor current versus the capacitor voltage yields the trajectory shown in Fig.

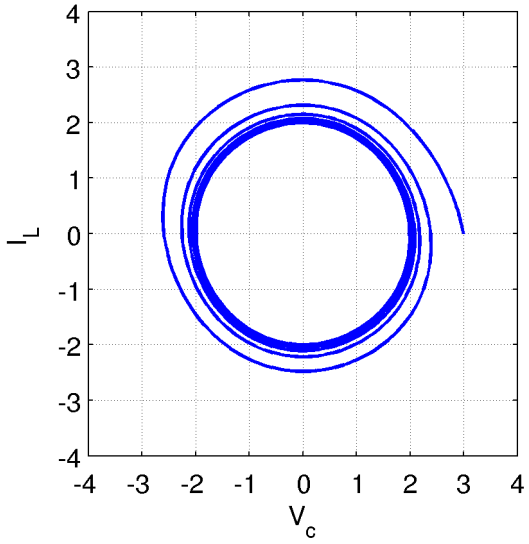

Figure 2.7: Inductor current vs. capacitor voltage for the Van der Pol oscillator (Netlist 2.3.1).

Restarting from previous results

As desired simulation starts from initial conditions set at $V_c = 3V$ and $I_L = 0$. It is sometimes useful to start a time domain simulation from the last point of a previous simulation. This can be done using option `restart` whose default value is `yes` as in the second analysis line:

```
Tran1 tran tstop=120 restart=no
```

setting `restart=no` instructs the simulator to initiate this time domain simulation using as initial point the last point computed in the immediately previous `tran`, `shooting` or `envelope` simulation. Note that, when restarting from a previous simulation, option `tstop` must take into

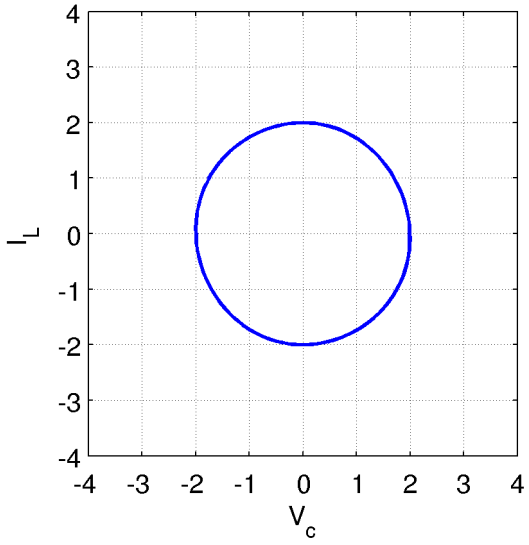

Figure 2.8: Inductor current vs. capacitor voltage for the Van der Pol oscillator (Netlist 2.3.1).

account the stop time of the previous simulation, in our case setting `tstop = 120` means that,

since Tran1 starts from time t=100s, total simulation time is 20s.

Solutions, i.e. "last points" of time domain analysis can also be saved and loaded using commands load <filename> and save <filename> respectively. Results are shown in Fig. 2.8, as it can be seen, since the previous analysis ends up very close to the oscillator "steady state" trajectory the circuit limit cycle is evidenced.

2.3.2 A simple linear LC oscillator and frequency warping effects

The circuit we consider now has the sole purpose of showing a little known effect on simulation of the integration method employed [BS03]. PAN default integration method is *trapezoidal*, but also the Gear methods up to order 6 are implemented. Consider the netlist in 2.3.2.

Netlist 2.3.2 — Linear LC oscillator. – File: lc.pan –

```
ground electrical 0

; Analyses
Tran0 tran tstop=1000 uic=1    ; default trapezoidal method
Tran1 tran tstop=1000 uic=1 method=2 order=2 ; Gear order 2
Tran2 tran tstop=1000 uic=1 method=2 order=6 ; Gear order 6

; Circuit
c1    n10     0  capacitor c=1 icon=3
l1    n10     0  inductor  l=1
```

Three time domain analyses are performed using three different integration methods. Results are quite interesting! All simulations start from the same initial conditions (see Fig. 2.9 left

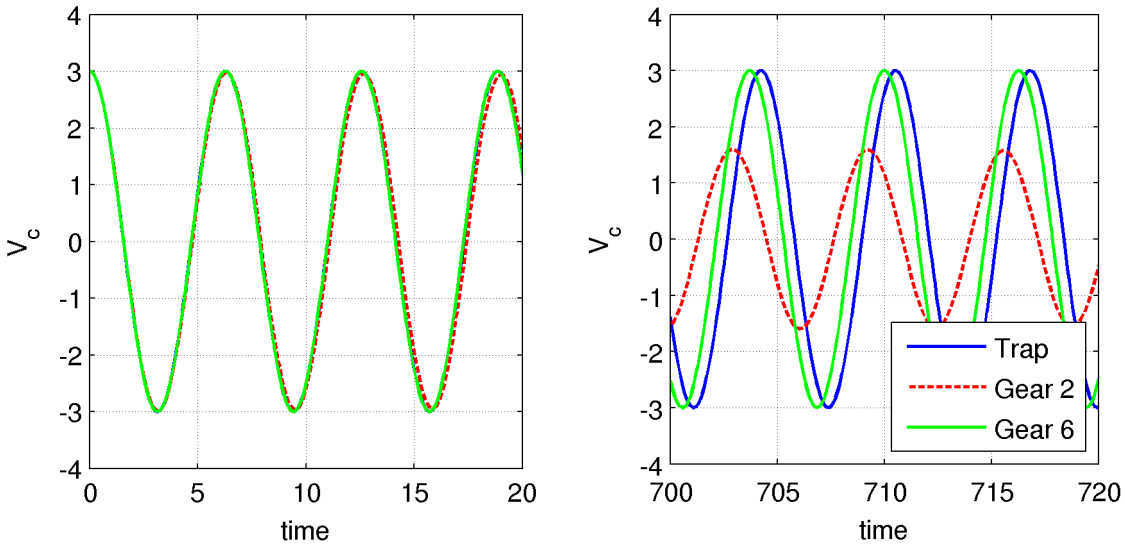

Figure 2.9: Capacitor voltage in the first 20s (left) and from 700s to 720s (right) for the simple LC oscillator (Netlist 2.3.2).

panel) but, even in the first few periods, show some differences. After about 100 periods (see Fig. 2.9 right panel), differences are dramatic. The LC oscillator is conservative, meaning that energy stored in the LC "tank" should remain constant in time. Nevertheless, the second order Gear algorithm slowly decays to zero. The sixth order Gear algorithm does not decay to zero

but displays a relevant phase shift with respect to the Trapezoid simulation. This phase shift is actually a different evaluation of frequency, details on this *frequency warping* effect (that is well known to astrophysicist) and its impact in circuit simulation can be found in [BS03].

The frequency warping effect can be evidenced by adding to all simulation lines above the option `findperiod=yes`. This activates a rather simple but effective period computation algorithm. Estimated oscillator periods found using the three different integration algorithms are significantly different as evidenced in Fig. 2.10. `PAN` 's time domain analysis has a large

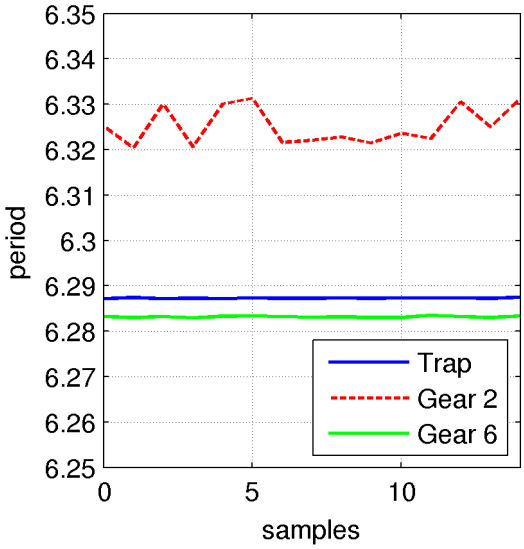

Figure 2.10: Oscillator period as evaluated by three different integration algorithms.

number of other options (more than 70!), as always a list and brief description of these options can be found with command `pan -s tran`.

2.4 Envelope time domain analysis: envelope

The Envelope time domain analysis performs a "fast" time domain analysis using information given by the envelope of the solution waveforms. Waveforms resulting from this analysis appear to "jump" in time rapidly converging towards a stable steady state solution. Results of the envelope analysis can be used to restart other time domain simulations and is useful to quickly arrive in proximity of a steady state solution.

2.4.1 The Bubba oscillator, using Schematic Capture and Netlisting tools

Consider now a simple sine wave oscillator, specifically a modified and simplified version of the Bubba oscillator found at page 17 of [MP01]. Since we do not have a netlist for this oscillator we will draw one using the standard open source tool `gschem` that is part of the `geda` tool suite. The schematic of the Bubba oscillator is shown in Fig. 2.11, note that a "SPICE include" has been added so that the model of the `TLV247xA` op-amp is included in the final spice netlist.

This schematic can be transformed in a spice netlist using `gnetlist`, another tool in the `geda` suite. This tool includes a number of different backends that can be used to process a schematic in many different ways. The spice backend that we will use is `spice-sdb` (the older backend `spice` is discontinued as far as we know). The command line that extracts from the schematic file (`bubba2.sch`) the netlist is:

```
prompt> gnetlist -g spice-sdb -o bubba2.pan -O include_mode -O sort-mode bubba2.pan
```

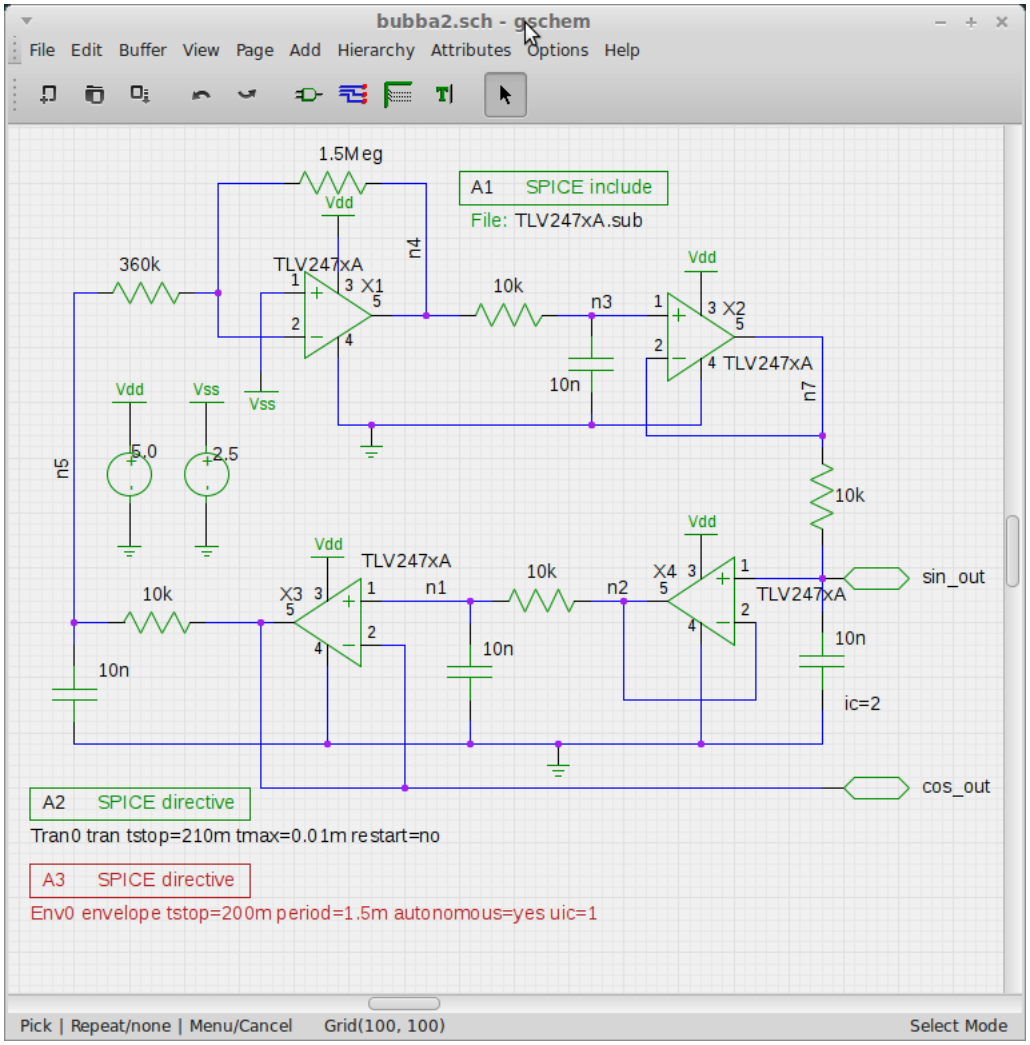

Figure 2.11: The Bubba oscillator in the geda tools `gschem` window (Netlist 2.4.1).

the line above instructs `gnetlist` to extract the netlist to file `bubba2.pan`.
Options `-O include_mode -O sort-mode` are for the backend and request the netlister to include external models without explicitly embedding them in the final netlist, and to write a netlist sorted by instance name. The resulting netlist is shown in 2.4.1[2]. As it can be seen the `SPICE directive` pseudocomponent has been used to include commands to execute simulations, another viable solution (possibly more efficient in many cases), is to include an external file containing all simulator directives and options or to directly edit the netlist file to add simulator directives as desired.
The final netlist is in spice syntax, but `PAN` automatically adjusts to read this format. Also most simulator directives can be given in spice format, but this feature is not completely supported and we suggest that native `PAN` syntax is used instead. Also note that the order of insertion of each single directive in the final netlist is not simple to control and, in some cases, this may lead to errors (e.g. when one simulation depends on the results of a previous one), if a strict ordering of analyses is required we suggest an external file containing simulation directives is used or, at least, to review and eventual correction of the final netlist file.

[2]Note that some quirk of the backend will put two identical `.INCLUDE` lines in the output netlist, you must delete or comment one of them in order to use the netlist.

Netlist 2.4.1 — The Bubba oscillator. – File: `bubba2.pan` –

```
* gnetlist -g spice-sdb -o bubba2.pan -O include_mode bubba2.sch
************************************************************
* Spice file generated by gnetlist                         *
* spice-sdb version 4.28.2007 by SDB --                    *
* provides advanced spice netlisting capability.           *
* Documentation at http://www.brorson.com/gEDA/SPICE/      *
************************************************************
.INCLUDE TLV247xA.sub
*==============  Begin SPICE netlist of main design ============
Env0 envelope tstop=200m period=1.5m autonomous=yes uic=1
Tran0 tran tstop=210m tmax=0.01m restart=no
Vss_s Vss 0 2.5
Vdd_s Vdd 0 5.0
C4 0 n5 10n
C3 0 n1 10n
C2 0 sin_out 10n  ic=2
R4 sin_out n7 10k
R6 n5 cos_out 10k
R1 n5 1 360k
R2 1 n4 1.5Meg
C1 0 n3 10n
R5 n1 n2 10k
R3 n4 n3 10k
X4 sin_out n2 Vdd 0 n2 TLV247xA
X2 n3 n7 Vdd 0 n7 TLV247xA
X3 n1 cos_out Vdd 0 cos_out TLV247xA
X1 Vss 1 Vdd 0 n4 TLV247xA
.end
```

Note that, in our case, a strict ordering *is* required, since we wish to perform a quick `envelope` simulation followed by a short *tran* analysis that restarts from the previous envelope results (note the `restart=no` option of analysis `Tran0`). The `Env0` simulation line, i.e.

```
Env0 envelope tstop=200m period=1.5m autonomous=yes uic=1
```

instructs `PAN` to perform an envelope analysis starting at time 0, ending at 200ns, with an approximate oscillator period of 1.5ms on an autonomous circuit (i.e. without external periodic inputs, different algorithms are used in the two cases), and, finally, using the initial conditions given in the netlist.
Results of this analysis are shown in Fig. 2.12. Envelope analysis only performs few cycles of the "fast dynamics" at a time predicting and simulating new cycles further in time only when needed. This is shown in the left panel. A brief transient analysis follows. Details of the last envelope sinusoids, followed by the initial transient ones are shown in the right panel.

2.5 Steady state time domain analysis: shooting

There are many situation where the designer is interested in finding the periodic steady state solution of a circuit. While a first and naive approach could be that of performing a long time domain simulation, waiting for the solution found to settle in a stable situation, this could lead to

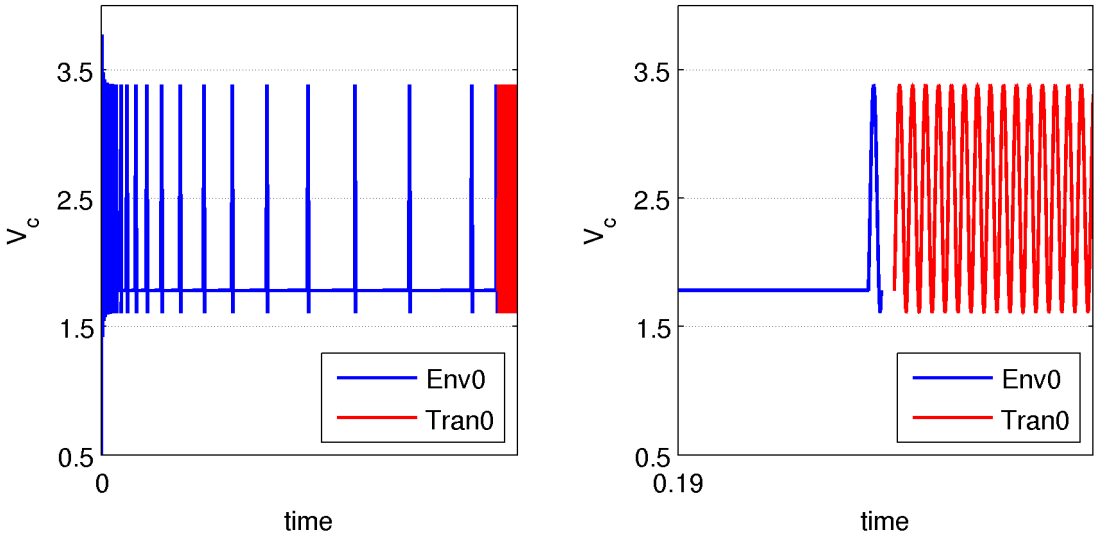

Figure 2.12: Envelope analysis for the bubba oscillator followed by a transient analysis, in the left panel the full simulation time is shown. The envelope anaysis, as expected, fully simulates the oscillator only in short time windows. In the right one a detail of the final time of the envelope analysis and start of the transient is shown.

very long simulation times for certain classes of circuits, such as high-Q oscillators. The shooting method overcomes this problem by solving a boundary value problem that forces a periodicity condition by finding an initial condition x_0 such that the circuit, after one period, is back at x_0. The circuit can be *autonomous* or *non-autonomous*, simply meaning, in the second case, that the state equations of the circuit have explicit dependence on time, or, from a practical point of view, that there are independent periodic voltage or current generators in the circuit. Since we are looking for a periodic steady state, in the *non-autonomous* case the circuit is *periodically* forced with known period T; on the other hand, if the circuit is autonomous, period T must also be determined. The shooting algorithm requires computing the sensitivity of the final solution with respect to the initial conditions[3]. This is quite straightforward if all equations are continuous, but requires some work otherwise.

2.5.1 The Butler overtone oscillator

The oscillator used in this section, known as the *Butler overtone oscillator*, is taken from [Rhe95] and is shown in Fig. 2.13. It is a crystal oscillator, the crystal is modelled as a simple third order RLC subcircuit that, in the `PAN` netlist shown below in 2.5.1, is named `xtal`. The bjt model for the `2N5179` device has been taken from the National Semiconductors site.

Netlist 2.5.1 — The Butler oscillator. – File: `butler.pan` –

```
ground electrical 0

options gground=1.0e-40 topcheck=2
parameters FREQ=120M Period=1/FREQ
```

[3]The method is called *shooting* for this reason: it is analogous to the adjustments that a gunman makes to his initial cannon tilt and yaw (i.e. the initial guess) in order to reach the target. In our case the gunman is actually "firing" a boomerang, expecting it to come back in the same point where it was thrown.

```
; Analyses
Dc dc print=yes

Tran tran tstop=500*Period uic=yes method=2 order=6 annotate=5 \
             tmax=Period/100

Envel envelope tstop=250k*0.55*Period method=1 restart=no \
                    autonomous=yes tmax=Period/1k period=Period

ShootGE4 shooting period=Period printnodes=yes uic=yes restart=no \
                autonomous=yes method=2 order=4 tmax=Period/1k \
                floquet=yes

; Circuit
q1       0     b    e    neg   Q2N5179
re       e   neg         resistor  r=560
rb       b  n200         resistor  r=33
c1    n200  loop         capacitor c=15p icon=20
c2    loop     0         capacitor c=47p
r2    n200     0         resistor  r=2.7k
l     n200  n300         inductor  l=140n
co    n300     0         capacitor c=1200p
r1    n300   neg         resistor  r=1.8k
lo    n400  loop         inductor  l=420.82496n
xtl   n400  loop         xtal
vn       0   neg         vsource vdc=10

vlo     e  n400         vsource vdc=0

define xtal (term1 term2)
    parameters RM=93
    co   term1    term2   capacitor c=4.18p
    rm   term1     n100   resistor  r=RM
    cm    n100     n200   capacitor c=0.00026p
    lm    n200    term2   inductor  l=6.76557m
end

model Q2N5179  bjt npn=yes is=69.28a xti=3 eg=1.11 vaf=100 bf=282.1 \
                   ne=1.177 ise=69.28E-18 ikf=22.03m xtb=1.5 br=1.176 \
                   nc=2 isc=0 rc=4 cjc=893.1f mjc=.3017 vjc=.75 fc=.5 \
                   cje=939.8f mje=.3453 vje=.75 tr=1.588n tf=141.1p \
                   itf=.27 vtf=10 xtf=30 rb=10
```

Note that several analyses are performed, a first `dc` analysis is followed by a first relatively short `tran` that, in turn, is followed by a relatively long `envelope` starting from the results of the previous `tran` and, finally, by a `shooting` analysis.
This oscillator is not easy to simulate since it requires a very large number of periods to power

up, meaning that a full tran analysis could be to expensive in terms of time to perform. On the other hand if a first short tran is not performed, the long envelope simulation that follows does not converge. After invoking the simulator a large number of log messages are output. The

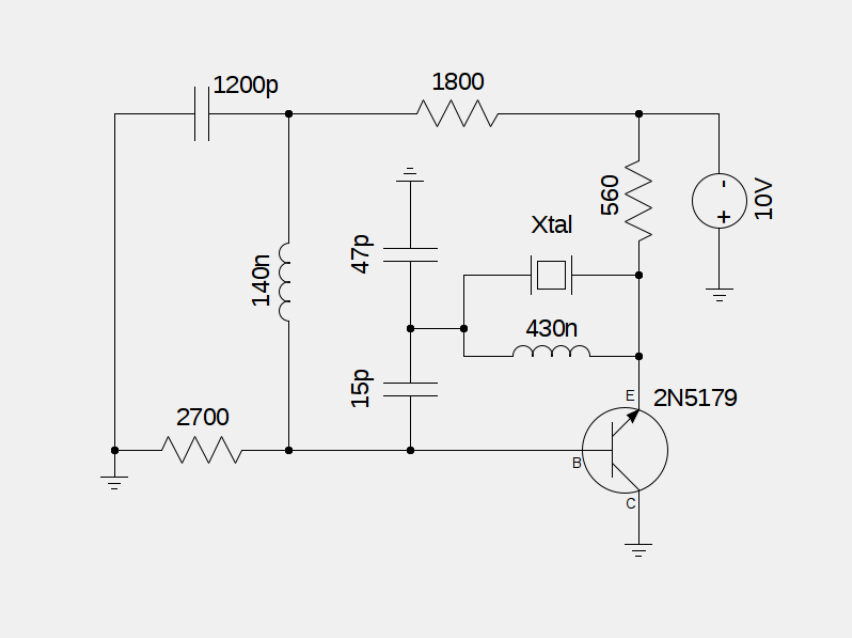

Figure 2.13: The Butler overtone oscillator.

largest part of the messages come from the envelope analysis, and are not interesting in this context. The most relevant messages are those relative to the last shooting analysis. A small portion of the log file is shown below in 2.5.2.

Logfile 2.5.2 — A snip of the logfile of the Butler Oscillator. – butler.log –

```
....
Notice given by pan during shooting analysis "ShootGE5".
    The relative error in identifying the Floquet multiplier theoretically
    equal to 1 is 20.872 u; the second multiplier gives a relative error
    equal to 35.319 u; the scaling factor is equal to 652.90 G.

Floquet multipliers with |real part| larger than 9.99998e-07:

    -5.3635e-01
     5.9922e-01
     8.3356e-01 +  3.4287e-02i   |0.83426|
     8.3356e-01 -  3.4287e-02i   |0.83426|
  *  9.9349e-01
     1.0000e+00
     9.9991e-01
     6.4138e-06

First Floquet exponent: 783.27 k
....
```

These messages are strictly related to the algorithm used to compute the shooting solution. Their meaning goes beyond the scope of this simple walk through and is discussed in detail in many papers and, more in general, in several books (see. e.g. [BGS09; BGS10; BMS05; Bra+10;

Far94; Kuz95]). Results of the `tran` and `envelope` analysis, clearly showing the very long

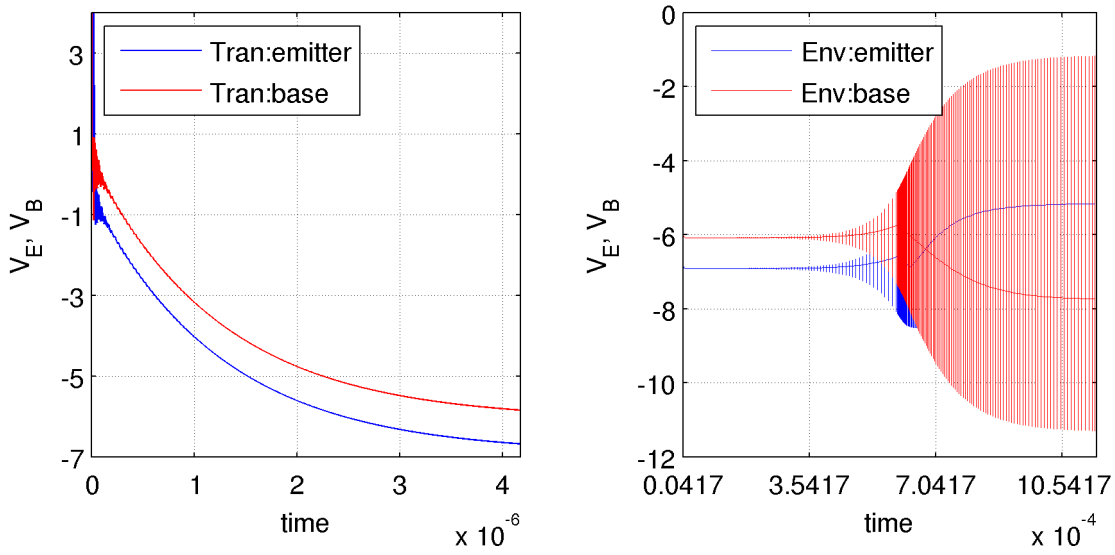

Figure 2.14: The first short `tran` analysis for the Butler oscillator (right panel) and the long `envelope` analysis that follows (left panel).

oscillator power up sequence, are depicted in Figure 2.14, the initial `tran` analysis in the left panel the long `envelope` analysis in the right panel. Details of this last analysis are shown in Figure 2.15. The `envelope` analysis arrives very close to the limit cycle (i.e. steady state) of the oscillator. The final `shooting` analysis computes one cycle of the steady state waveform. Results are shown in Figure 2.16.

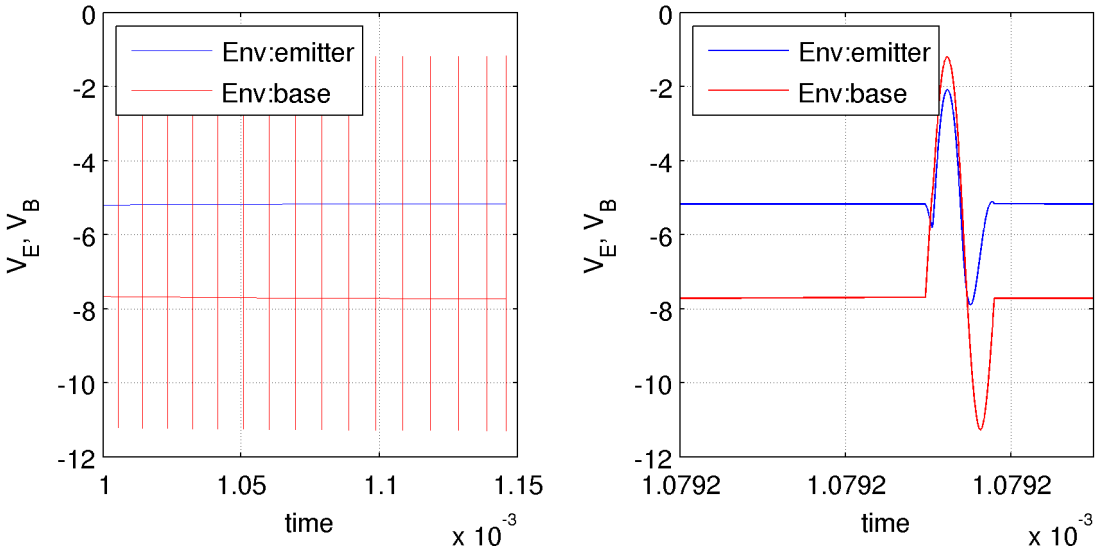

Figure 2.15: Details of the long `envelope` analysis for the Butler oscillator towards the end of the simulation.

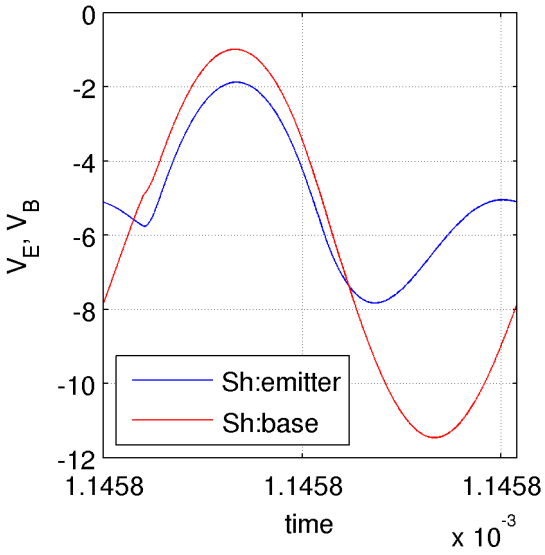

Figure 2.16: Final `shooting` analysis for the Butler oscillator.

3. AMS simulation with PAN

Modern circuit design is often based on the Analog Mixed Signal (AMS) "paradigma", where various parts of the overall system are modeled using a different approach. The most common situation is where an analog circuit, such as the ones seen in the previous section, are connected with *digital* circuits or to blocks described using behavioural languages.
PAN is capable of simulating AMS circuits seamlessly integrating the analog netlist with a Verilog description of the digital part and a Verilog-A description of the behavioural part.

Simple analysis are available, but time-domain steady state or more complex analysis such as PAC pr PNOISE are also available. This is not common, since "standard" steady state algorithms require continuous vector fields, and this is not the case of AMS circuits.

In the following a few simple examples, mostly derived from those published by the authors in [Biz+] are shown from an operative point of view. Please refer to the cited paper and its bibliography for technical details concerning the used algorithms.

3.1 A simple Analog Digital oscillator

The first circuit we will consider is a very simple Pulse Energy Restore Oscillator (PERO) [BBS11; Biz+11]. The schematic of the PERO oscillator is shown in Fig. 3.1. It is composed by

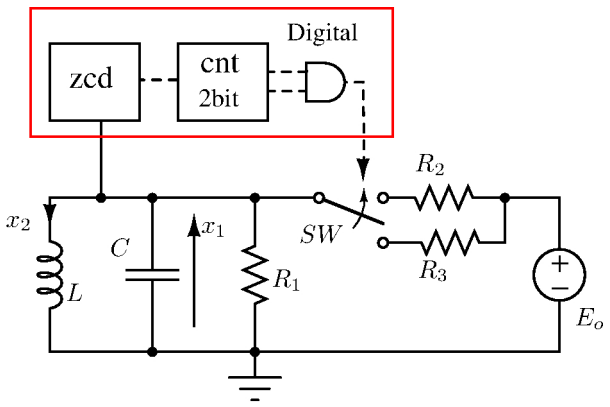

Figure 3.1: The schematic of the PERO oscillator. $C = 1\,\mathrm{F}$, $L = 1\,\mathrm{H}$, $R_1 = 10\,\Omega$, $R_2 = 100\,\mathrm{M}\Omega$, $R_3 = 5\,\Omega$, $E_o = 1\,\mathrm{V}$.

a linear analog part and a digital Finite State Machine that recharges the energy in the LC tank. A zero crossing detector (zcd) is used as input to a 2 bit digital counter (cnt), that closes switch S when both its bits are high.

The analog netlist is written in standard PAN form and is shown in Netlist 3.1.1

Netlist 3.1.1 — The PERO oscillator (analog part). – File: pero.pan –

```
ground electrical  gnd

parameters VDD=1

verilog_include comb.v

Tran tran tstop=300 uic=yes
Sh0 shooting period=25 autonomous=yes restart=no \
             method=2 order=2 solver=2 minper=19 floquet=yes

Pnoise pnoise start=10u stop=100m dec=4000 onodes=["x"] sweeptype=0

r1    x   gnd   resistor    r=10
c1    x   gnd   capacitor   c=1 ic=1
l1    x   gnd   inductor    l=1
sw    x   vdd   y  gnd  SW
vr  vdd   gnd   vsource     vdc=VDD

model SW vswitch ron=5  roff=100M von=0.6*VDD voff=0.4*VDD

A2d1  x   gnd   a2d  dignet="cnt.X" vl=0 vh=0
D2a1  y   gnd   d2a  dignet="cnt.Y" vl=0 vh=VDD
D2a2  z   gnd   d2a  dignet="cnt.Count" vl=0 vh=VDD
```

Voltage controlled switch sw implements also resistors R_2 and R_3 of the netlist; in fact

```
model SW vswitch ron=5 roff=100M von=0.6*VDD voff=0.4*VDD
```

sets the values of roff and ron to those chosen, respectively, for R_2 and R_3.
The AMS specific parts of this netlist are two, the first is the single line

```
verilog_include comb.v
```

that instructs the simulator to read also a Verilog file called, in this case, comb.v (more than one file can be read if needed). This file, shown below in Netlist 3.1.2, contains the description of the digital part of the circuit. The second AMS specific part is used to connect the analog part to the digital one.

```
A2d1  x   gnd   a2d  dignet="cnt.X" vl=0 vh=0
D2a1  y   gnd   d2a  dignet="cnt.Y" vl=0 vh=VDD
D2a2  z   gnd   d2a  dignet="cnt.Count" vl=0 vh=VDD
```

These three line define three *pseudo* components that convert analog signals to digital or viceversa (respectively a2d and d2a).
Node names are given in standard PAN netlist syntax, while the corresponding digital net by using the dignet keyword. Note that the format for digital net is *module.NET*. Voltage levels

must also be stated, representing a threshold level for the a2d components, and mapping levels for the d2a ones.
The digital part of the circuit is quite simple and written is standard Verilog language.

Netlist 3.1.2 — The PERO oscillator (digital part). – File: comb.v –

```
‘timescale 1ms/10us

module cnt;
reg X, Y;
reg [1:0] Count;

initial begin
    Count = 0;
    Y     = 1;
end

always @(posedge X) begin
//    $display( "Time: ", $realtime );
    Count[0] = ! Count[0];
    Count[1] = (!Count[0]) ^ Count[1];
end

always @(Count) begin
    if( Count[0] && Count[1] )
Y = 1;
    else
Y = 0;
end

endmodule
```

The simulation is executed exactly as any other analog only circuit,

```
prompt> pan pero.pan
```

and the usual directory pero.raw containing results for each analysis is created.

Let us first see results from the transient analysis Tran. The two analog state variables, i.e. capacitor voltage and inductor current are represented in the netlist file by node voltage x and current l1:i respectively. The third state variable is digital and correspond to the 2 bit register Count defined in the Verilog part of the netlist. This register is available as a digital net (keyword dignet) in the analog part of the netlist as cnt.Count, where cnt is the Verilog module name. This digital net is remapped to the analog net z by the d2a converter D2a2. The conversion transforms the two bit value to a single scalar value in the range defined by vl=0 vh=VDD, i.e., in our case, to the 4 possible values $0, 1/3, 2/3, 1$. The transient behaviour of these state variables is shown in the three panels of Fig. 3.2.

Following the transient analysis a shooting analysis is performed. This analysis finds the steady state solution for the circuit. Note that, since this circuit is autonomous, also its period must be determined and option autonomous = yes must be explicitly specified. The steady state behaviour of the circuit is shown in Fig. 3.3, where the state variables are shown as a trajectory in state space (on the z axis we have put, in this case, the binary counter value), and Fig. 3.4, where they are shown versus time.

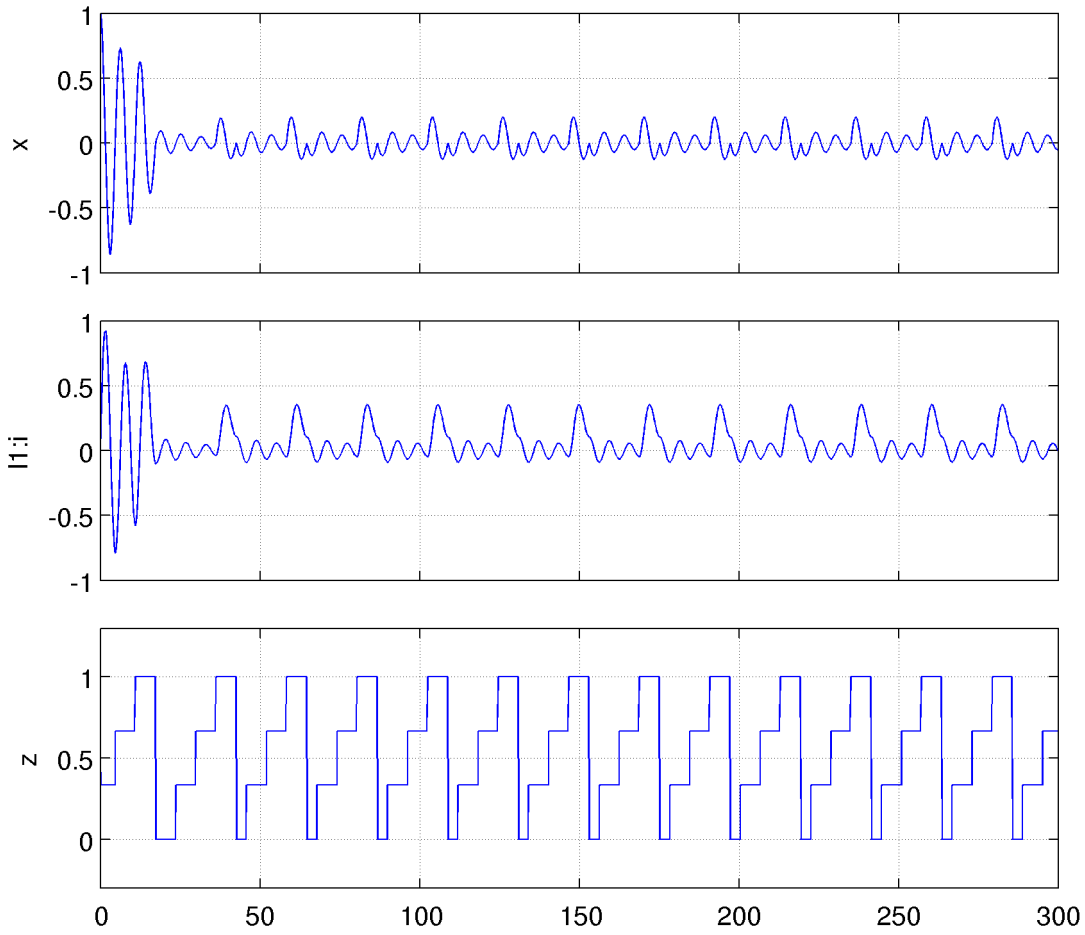

Figure 3.2: The time evolution of the three state variables of the PERO oscillator. Capacitor voltage is `x` in the top panel, inductor current `I1:i` in the centre panel, and the analog signal corresponding to the digital two bit counter value (remapped in the 0 to 1 range) in the bottom one.

3.2 The `pnoise` analysis

In the previous example, based on the PERO oscillator, after the shooting analysis a `pnoise` analysis is also performed. The relevant line is

```
Pnoise pnoise start=10u stop=100m dec=4000 onodes=["x"] sweeptype=0
```

This analysis requires a previous shooting analysis and computes the noise at a specified output (in our case node `x`) of circuits that admit a periodic solution.

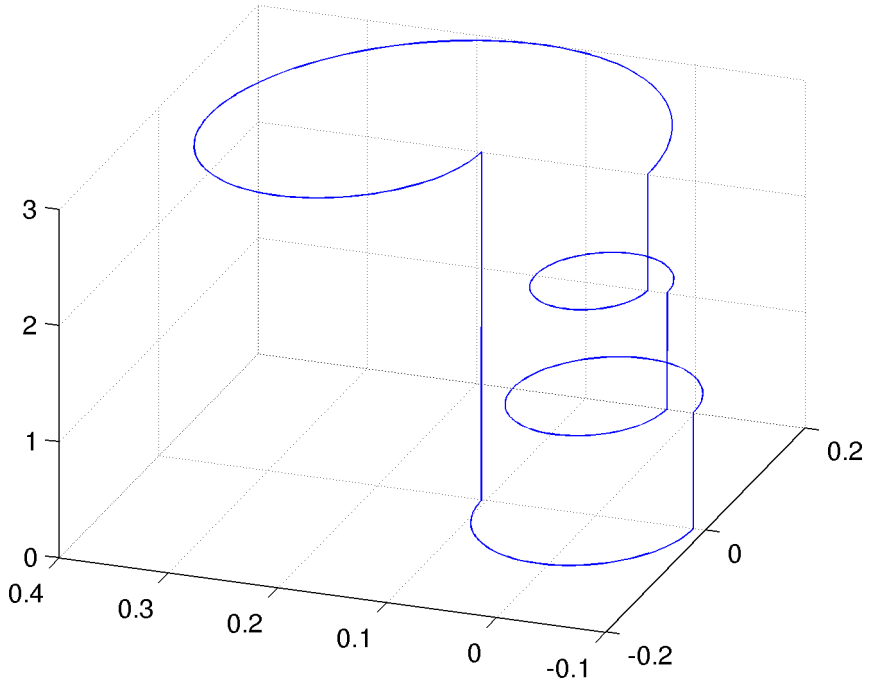

Figure 3.3: Steady state trajectory of the PERO oscillator plotted in state space.

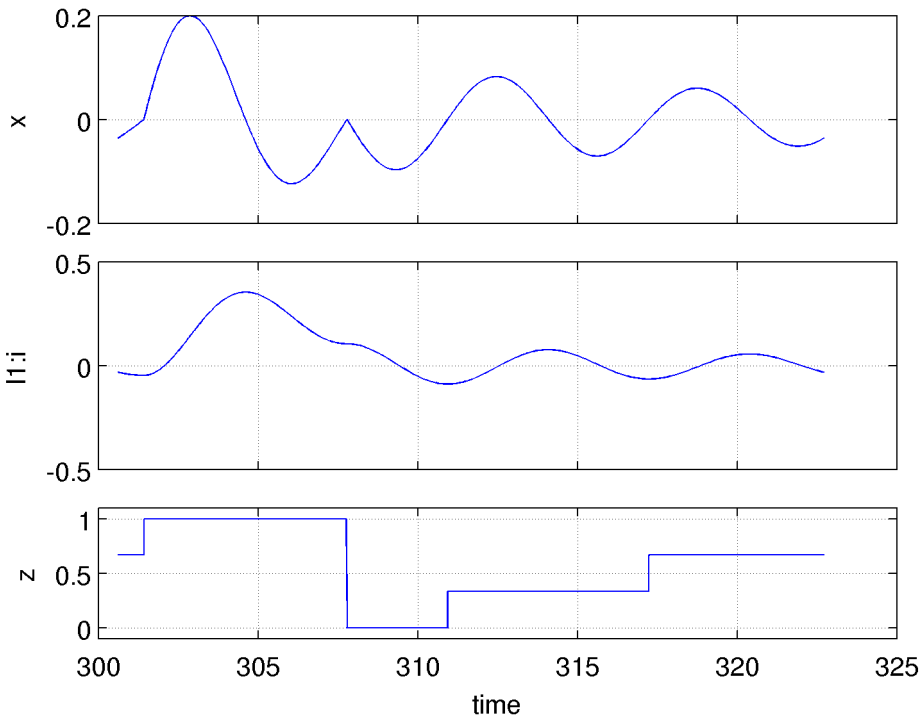

Figure 3.4: Steady state trajectory of the PERO oscillator plotted vs time. Note that, since the shooting analysis is restarted from the previous transient one, the time axis starts at 300.

4. Reference

4.1 Where and How: downloading and installing

The PAN circuit simulator binary can be downloaded from http://brambilla.ws.dei.polimi.it. The PAN binary requires a standard linux installation. PAN developers use Debian based systems (most recently Debian, Mint and Ubuntu), but PAN has been used without any pain with other Linux flavours, including RedHat, Gentoo, Sabayon and several others.

A Linux live CD image including (possibly not) the latest version of the simulator and of the demo files is also available. This allows quick evaluation without actually installing anything in your computer.

Using PAN is relatively simple, and only requires knowledge of basic Linux terminal commands and the use of any ASCII editor (such as Vi). As it is netlists are entered in ASCII format, but, as it will be shown in the sequel, thanks to the high compatibility of PAN with well known simulator languages, almost any standard netlister can be used, as long as some adjustments to the command language are made.

A quick reference is available with

```
pan -s
```

a list of keywords appears, and further help on a specific topic can be obtained with the command

```
pan -s topic
```

In the following sections a quick reference for all devices and analyses available in PAN is provided. In most cases these are identical to the inline quick reference.

4.2 Basic one-port device

4.2.1 Linear Two Terminal Resistor

The resistor is tied between the n1 and n2 nodes; a possible third terminal is for a thermal connection and it has meaning only during electro-thermal analyses. During electro-thermal simulations the self-heating effects of the resistor are considered if its thermal node is connected to the thermal network. The ‘r’ parameter is the resistance in ohms; it can be either positive or negative but it must not be zero. Two types of resistors are admitted: linear and piece-wise linear. The piece-wise linear resistor is described by giving the coordinates of the extremes of the segments composing the characteristic in the voltage-current plane. The piece-wise characteristic is continued adove and below the given extremes with a slope equal to that of the correspoding

last extreme. Orizontal segments (zero conducance) are allowed but vertical ones are not. To aid convergence of numerical algorithms a parasitic resistor of conductance <gmin> is connected in parallel to each pwl resistor.

ELEMENT DEFINITION:
Name n1 n2 [t1] **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:

`r` Real (-): resistance value.

`g` Real (-): conductance value.

`m` Integer (1): number of identical resistors connected in parallel.

`func` String (-): the algebraic function that describes the 'r(v)' resistance/voltage characteristic of the resistor.

`l` Real (0.0): resistor length.

`w` Real (1.0e-5): resistor width.

`tc` Real (0): coefficient of the linear term in the temperature dependence expression of resistance. Temperature affects only linear resistors. If a resistor model is associated to the instance and this parameter is specified only in the model, the instance inherits the value given in the model.

`tc1` Real (0): coefficient of the linear term in the temperature dependence expression of resistance. Temperature affects only linear resistors. If a resistor model is associated to the instance and this parameter is specified only in the model, the instance inherits the value given in the model.

`tc2` Real (0): coefficient of the quadratic term in the temperature dependence expression of resistance. Temperature affects only linear resistors. If a resistor model is associated to the instance and this parameter is specified only in the model, the instance inherits the value given in the model.

`noisetype` Integer (5): types of generated noise; possible choices are:
0x0 : noiseless resistor; 0x1 : thermal noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

MODEL DEFINITION:

model *ModelName* **resistor** *<parameter=value> ...*

MODEL PARAMETERS:

`r` Real (1.0): the value of resistance is applied to all resistors referring to this model; the resistance value of each resistor instance overwrites the value given in the model.

`type` Resistor types:
1: linear resistor;
2: piece-wise linear resistor, specified by giving the voltage/current corners of its characteristic (see <v> and <i> parameters for more details.

`i` Current value at a corner of the piece-wise linear characteristic. This parameter has meaning only if the resistor is a piece-wise linear one.

`v` Voltage value at a corner of the piece-wise linear characteristic. This parameter has meaning only if the resistor is a piece-wise linear one.

`interp` The number of samples for characteristic interpolation. This parameter has meaning only for piece-wise linear resistor types. It acts at the corners of the pwl characteristic by smoothing it through the adoption of B-spline. It is suggested to check if the resulting characteristic is adequate.

`scalev` Real (1.0): scale voltages of a pwl characteristic.
`scalei` Real (1.0): scale currents of the pwl characteristic.
`swapiv` Boolean (false): when set, the two columns of the file describing the characteristic of a pwl resistor are swaped. Recall that as default the first column represents voltage values and the second one current values.
`pwlfile` String (-): the name of the file containig the (v,i) values specifing the characteristic of a pwl resistor. Instead of describing the characteristic through the <v> and <i> parameters it can be read more conveniently from a file. The format of data is: one point of the characteristic per row, each point composed of two real numbers separated by a space, the first represents voltage and the second the corresponding current.
`quiet` Boolean (false): it is a boolean parameter that when turned on suppreses the message about the file system path from which the file containing the pwl characteristic of the resistor is loaded. The simulator searches and opens files by sequentially looking in the directory list specified through the environment variable <PAN_PATH>. It opens the first file found in the directory list. This can generate errors if files with the same name are contained in different directories. For this reason a warning is given each time a file is found and opened.

`tc1` Coefficient of the linear term in the temperature dependence expression of resistance. Temperature affects only linear resistors.
`tc2` Coefficient of the quadratic term in the temperature dependence expression of resistance. Temperature affects only linear resistors.
`tnom` Nominal temperature at which resistance was determined.

`rsh` Real (0.0): sheet resistance.
`narrow` Real (0.0): narrowing of resistance.
`etch` Real (0): width narrowing due to etch per side.
`short` Real (0.0): shortening of resistance.
`etchl` Real (0): length narrowing due to etch per side.
`w` Real (1.0e-5): resistor with.
`l` Real (1.0e-5): resistor length.

`af` Real (1.0): flicker noise exponent.
`kf` Real (0.0): flicker noise coefficient.

4.2.2 Linear Two Terminal Capacitor

The capacitor is tied between n1 and n2. C is the capacitance in farads, and is a modifiable parameter

ELEMENT DEFINITION:
Name n1 n2 **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:
`c` Real (DefaultCapacitace): capacitance value.
`m` Integer (1): number of identical capacitors connected in parallel.
`w` Real (-): capacitor width.
`l` Real (-): capacitor length.
`scale` Real (1.0): capacitor scale factor.

`qv` String (-): the algebraic function that describes the 'q(v)' charge/voltage characteristic of the capacitor.
`cv` String (-): the algebraic function that describes the 'c(v)' capacitance/voltage characteristic of the capacitor.
`icon` Initial capacitor voltage. It can be used as initial condition by time domain algorithms like <tran>, <shooting>, <envelope> and <mtshooting>.
`ic` Initial capacitor voltage. It can be used as initial condition by time domain algorithms like <tran>, <shooting>, <envelope> and <mtshooting>.
`tc1` Real (0.0): first order temperature coefficient.
`tc2` Real (0.0): second order temperature coefficient.
MODEL DEFINITION:
model *ModelName* **capacitor** <*parameter=value*> ...

MODEL PARAMETERS:
`c` Real (DefaultCapacitace): capacitance value.
`c1, c1, ...` Real (0.0): the coefficients describing the characteristic of a polynomial capacitor. The charge Q is expressed as

$$Q(v) = C * v * (c0 + 1/2 * c1 * v + 1/3 * c2 * v^2 + ...)$$

where 'ck' is the k-th entry of the list. Note that the default value of 'c0' is 1.

`w` Real (-): capacitor width.
`l` Real (-): capacitor length.
`narrow` Real (1.0): width correcting factor.
`short` Real (1.0): length correcting factor.

`cj` Real (-): bottom capacitance per area.
`cjsw` Real (-): sidewall capacitance per meter.
`di` Real (3.9): relative dielectric constant.
`thick` Real (-): dielectric thickness.

`tc1` Real (0.0): first order temperature coefficient.
`tc2` Real (0.0): second order temperature coefficient.
`tnom` Real (-): temperature at which the capacitor parameters have been defined.

4.2.3 Two Terminal Inductor

The inductor is tied between the n1 and n2 terminals. Inductor can be linear or nonlinear; there are three different ways to describe a nonlinear inductor. Note that a nonlinear inductor can be described only through a model card specification. In the first case the nonlinear inductor is saturable and current controlled. It is modeled by the relation

$$Phi = SNBmaxtanh(NI/(LHmax)) + SNmu0I/L$$

where Phi is flux, S cross section of the core, N number of turns, L the average length of the core, Hmax value of magnetic field at saturation, Bmax value of the induction magnetic field at Hmax, mu0 the vacuum permittivity and finally I inductor branch current. As can be seen from the above formula nonlinearity is modeled by the 'tanh()' function. When 'N I / L > Hmax', the flux can be considered linearly dependent on I. The second mode to model a nonlinear current controlled inductor is through the polynomial

$$Phi = c0I + c1I^2 + c2I^3 + c3I^4 + c4I^5 + c5I^6$$

where c0, c1, c2, c3, c4 and c5 constant coefficients are specified in the model card; if omitted, default value of these parameters is 0. The third mode is similar to the second one but the nonlinear inductor is flux controlled

$$I = c0Phi + c1Phi^2 + c2Phi^3 + c3Phi^4 + c4Phi^5 + c5Phi^6$$

ELEMENT DEFINITION:
Name n1 n2 **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`l` Inductance. It is considered only in a linear inductor.
`icon` Initial inductor current
`ic` Initial inductor current
`fi` String (-): the algebraic function that describes the 'f(v)' flux/current characteristic of the inductor.
MODEL DEFINITION:
model *ModelName* **inductor** <*parameter*=*value*> ...

MODEL PARAMETERS:
`type` Inductor type: 1: linear inductor; 2: nonlinear saturable inductor; 3: nonlinear current controlled inductor; 4: nonlinear flux controlled inductor

---

`bmax` Maximum induction magnetic field (tesla) at saturation. It sets the value of the magnetic field when H = N * I / L is equal to Hmax, where N is the number of turns, I is inductor current and L is core length. It is meaningful only for a saturable inductor
`corearea` Magnetic core area (meter * meter). It is the area of the cross-section of the core and is meaningful only for a nonlinear saturable inductor
`corelength` Magnetic core length (meter). It is the average length of core and is meaningful only for a nonlinear saturable inductor
`hmax` Maximum magnetic field (amper / meter). When the magnetic field is above this value the inductor is considered saturated. It is meaningful only for a saturable inductor
`turns` Number of turns. It is meaningful only for a nonlinear saturable inductor

---

`c0` Coefficient of the modeling polynomial of nonlinear inductor.
`c1` Coefficient of the modeling polynomial of nonlinear inductor.
`c2` Coefficient of the modeling polynomial of nonlinear inductor.
`c3` Coefficient of the modeling polynomial of nonlinear inductor.
`c4` Coefficient of the modeling polynomial of nonlinear inductor.
`c5` Coefficient of the modeling polynomial of nonlinear inductor.

4.2.4 Independent Voltage Source

The positive node is n+ and the negative node is n-. Current through the source is computed and is defined to be positive if it flows from the positive node through the source to the negative node. The only meaningful parameter in a DC analysis is vdc. In an AC analysis only mag and phase are meaningful. In a harmonic analysis, vdc, mag1, mag2, phase1, phase2, etc are used. In a spectral analysis, vdc, mag1f, phase1f, mag2f and phase2f are used. In the time domain analyses, such as transient, shooting, envelope, multishooting, the piece-wise linear source, pulse

source, sinusoidal and exponential ones are meaningfull. In a periodic small signal analysis (pac analysis) only the xacmag and xacphase parameters are used. DC voltage and magnitudes are given in volts and phase is given in degrees.

ELEMENT DEFINITION:
Name n+ n- **vsource** *<parameter=value> ...*

ELEMENT PARAMETERS:

`vdc` Real (0.0): dc voltage.
`dc` Real (0.0): dc voltage.
`stepping` Boolean (true): enables or disable the source stepping continuation method during harmonic, spectra and dc analyses.
`type` String (-): the voltage source type.
`mag1, mag2, ...` Real/s (0.0): magnitude of fundamental and harmonics in harmonic analysis. As many harmonics as necessary can be used, that is, the number of harmonics is not upper limited.
`ph1, ph2, ...` Real/s (0.0): phase of fundamental and harmonics in harmonic analysis. As many harmonics as necessary can be used, that is, the number of harmonics is not upper limited.
`mag` Real (0.0): signal magnitude in small signal analysis (AC analysis).
`phase` Real (0.0): signal phase in small signal analysis (AC analysis).
`pacmag` Real (0.0): magnitude of the source in small signal periodic analysis (pac analysis).
`pacphase` Real (0.0): phase of the source in small signal periodic analysis (pac analysis).
`mag1t` Real (0.0): magnitude of first fundamental in spectral analysis.
`mag2t` Real (0.0): magnitude of second tone generated in the spectral analysis.
`freq2t` List [-]: the list of indices that defines the frequency of the second tone as a linear combination of the two fundamentals of the spectral analyses. It is composed of two items. The fist is the index the the harmonic of the first fundamantal and the second is the index of the harmonic of the second fundamental. For example 'freq2t=[1,-1]' defines the frequency (1*f1 -1*f2).
`phase1t` Real (0.0): phase of first fundamental in spectral analysis.
`phase2t` Real (0.0): phase of second time in the spectral analysis.
`freq` Real (0.0): frequency of the sinusoidal voltage source in time domain analyses such as tran, shooring, envelope and mtshooting.
`vsin` Real (0.0): maximum voltage of the sinusoidal source in time domain analyses such as tran, shooring, envelope and mtshooting.
`ampl` Real (0.0): maximum voltage of the sinusoidal source in time domain analyses such as tran, shooring, envelope and mtshooting.
`vphase` Degree (0.0): phase of the sinusoidal source in time domain analyses such as tran, shooring, envelope and mtshooting.
`tdsin` Real (0.0): the time interval along which the sinusoidal source is kept off. The first period of the source beging exactly after this time interval, more rigorously the generated waveform is 'v(t)=vsin * sin(omega*(t-tdsin)+vphase) * exp(damping*(t-tdsin))'.
`damping` Real (0.0): the damping factor of the sinusoidal source 'v(t)=vsin * sin(omega*(t-tdsin)+vphase) * exp(damping*(t-tdsin))'.
`v` Real (0.0): voltage value of a point in the pwl time sequence. It is employed only in time domain analyses.
`t` Real (0.0): time instant of a point in the pwl sequence. It is employed only in time domain analyses. If the simulation time stop is set after the last time point of the pwl sequence,

the entire sequence is repeated as a periodic signal

`tr` Real (0.0): rising time of the pulse generator.

`trise` Real (0.0): rising time of the pulse generator.

`rise` Real (0.0): rising time of the pulse generator.

`tf` Real (0.0): falling time of the pulse generator.

`fall` Real (0.0): falling time of the pulse generator.

`tfall` Real (0.0): falling time of the pulse generator.

`v1` Real (0.0): voltage value of the pulse waveform in the time interval outside that specified by the duty cycle.

`val0` Real (0.0): voltage value of the pulse waveform in the time interval outside that specified by the duty cycle.

`v2` Real (0.0): voltage value of the pulse waveform in the time interval specified by the duty cycle.

`val1` Real (0.0): voltage value of the pulse waveform in the time interval specified by the duty cycle.

`td` Real (0.0): time delay before the first rising front in the waveform generated by the pulse generator.

`width` Real (0.0): duty cycle of the pulse waveform. It must be an absolute time interval and not a relative value with respect to the period of the pulse generator.

`period` Real (0.0): the period of the pulse and pwl generator. If not specified for a pwl generator then the waveform is continued above the last time point by maintaining the last voltage value (it is continues as a constant waveform).

`ev` The voltage of the expotential source.

`td1` Delay time in the application of the voltage rise by the exponential voltage source.

`tau1` Rise time constant of the exponential voltage source.

`td2` Delay time in the application of the voltage fall by the exponential voltage source.

`tau2` Fall time constant of the exponential voltage source.

`fmfreq` Real (0.0): frequency of the phase modulating signal. Phase modulation is active only in time domain analyses such as tran, envelope, shooting and mtshooting.

`fmmag` Real (0.0): maximum value of the phase modulating signal. Phase modulation is active only in time domain analyses such as tran, envelope, shooting and mtshooting. The frequency modulation acts on the sinusoidal signal generation that must be thus contestually specified as part of the same voltage source.

`fmphase` Real (0.0): phase of the phase modulating signal. Phase modulation is active only in time domain analyses such as tran, envelope, shooting and mtshooting.

4.2.5 Independent Current Source

Positive current exits the source node and enters the sink node. The only meaningful parameter in a DC analysis is idc. In an AC analysis only mag and phase are meaningful. In a harmonic analysis, idc, mag1, mag2, phase1, phase2, etc are used. In a spectral analysis, idc, mag1t, phase1t, mag2t and phase2t are used. In the time domain analyses, such as transient, shooting, envelope, multishooting, the piece-wise linear source, pulse source, sinusoidal and exponential ones are meaningfull. In a periodic small signal analysis (pac analysis) only the xacmag and xacphase parameters are used. DC current and magnitudes are given in ampere and phase is given in degrees.

ELEMENT DEFINITION:
Name sink source **isource** *<parameter=value> ...*

ELEMENT PARAMETERS:

`dc` Real (0.0): DC current.
`idc` Real (0.0): DC current.
`stepping` Enables or disable the source stepping (default enabled).
`type` String (-): the current source type.
`freq` Frequency of the current source.
`isin` Maximum current of the sinusoidal source.
`iphase` Real (0): phase (degree) of the sinusoidal source. It is considered only in time domain analyses.
`tdsin` Real (0.0): the time interval along which the sinusoidal source is kept off. The first period of the source beging exactly after this time interval, more rigorously the generated waveform is 'i(t)=isin * sin(omega*(t-tdsin)+vphase) * exp(damping*(t-tdsin))'.
`damping` Real (0.0): the damping factor of the sinusoidal source 'i(t)=isin * sin(omega*(t-tdsin)+vphase) * exp(damping*(t-tdsin))'.
`mag1, mag2, ...` Magnitude of desired fundamental and harmonics. These is no upper limit to the number of harmonics.
`ph1, ph2, ...` Phase of desired fundamental and harmonics. These is no upper limit to the number of harmonics.
`mag` Small signal magnitude.
`phase` Small signal phase.
`pacmag` Small signal magnitude in periodic small signal analysis (pss analysis).
`pacphase` Real (0.0): phase of the source in small signal periodic analysis (pac analysis).
`mag1t` Real (0.0): magnitude of first fundamental in spectral analysis.
`mag2t` Real (0.0): magnitude of second tone generated in the spectral analysis.
`freq2t` List [-]: the list of indices that defines the frequency of the second tone as a linear combination of the two fundamentals of the spectral analyses. It is composed of two items. The fist one is the index of the harmonic of the first fundamantal and the second is the index of the harmonic of the second fundamental. For example 'freq2t=[1,-1]' defines the frequency (1*f1 -1*f2).
`phase1t` Real (0.0): phase of second time in the spectral analysis.
`phase2t` Real (0.0): phase of second time in the spectral analysis.
`i` Current corner of the pwl time function
`t` Time instant of a corner in the pwl function. Pwl sequence is considered only in time domain analyses.
`tr` Rising time of the pulse waveform.
`tf` Falling time of the pulse waveform.
`i2` Current value during the up portion of the pulse waveform. Note that this value may be negative.
`i1` Current value during the low portion of the pulse waveform.
`td` Real (0.0): time delay before the first rising front in the waveform generated by the pulse generator or the time delay before the beginning of the first period of the sinusoidal waveform generated by the sinusoidal generator.
`width` Duty cycle of the pulse waveform. It must be expresse as an absolute time interval and not as a relative number with respect to the period of the pulse waveform.
`period` Real (0.0): the period of the pulse and pwl generator. If not specified for a pwl generator then the waveform is continued above the last time point by maintaining the last voltage value (it is continues as a constant waveform).

4.3 Basic two-port device

4.3.1 Voltage Controlled Current Source

Voltage controlled current source is a device with at least 2 terminals; current exits the source terminal and enters the sink terminal. The number of terminals must be even, those above the first 2 are the controlling terminals. There is a <gm> parameter for each controlling pair that linearly relates the output current to the voltage across the corresponding input port. External macromodels are supported. The routine describing the macromodel is called with the following ordered sequence of arguments:

<input> int : an index that uniquely identifies the instance of the VCCS. This index starts from 0 and it is incremented by 1 each time a new instance is found in the netlist; <input> int : number of ports of this VCCS <input> double *: the array of the port voltages <input> int : number of state variable <input> double**: the two vectors of state variable values at the current n time point (index 0) and at the previous (n-1) time point (index 1). In other words, the array can be indexed as 'array[0][StateVarNum]' current time point and 'array[1][StateVarNum]' previous time point. <output> double *: the array of the transconductances (the meaning is the same as that of the <gm> parameter) <output> double *: the value of the current flowing through the output port <input> double : the value of simulation time. It has meaning only in time domain analyses.

The possible routine handling parameters of the macromodel is called with the arguments:

<input> int : an index that uniquely identifies the instance of the VCCS. <input> char * : parameter name; <input> double : parameter value.

It must return 1 upon successful execution and 0 otherwise.

ELEMENT DEFINITION:
Name sink source <a lot of terminals> **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:

`gain1, gain2, ...` Real (0.0): gain of each driving port of the VCCS; since there can be more than one input port, there must be as much of these parameters as the input ports. This applies only if the VCCS is not modelled by an external macromodel. In this case gains are specified by the macromodel.

`gm` Real (-): transconductance. This parameter is equivalent to <gain1>.

`delay` Real (0.0): time delay.

`func` The function that describes the characteristic of the VCCS.

`offset` Real (0): the offset current generated by the VCCS. This current is added to that due to the driving signals.

`digital` Boolean (false): tells that this is a digital macromodel. The term 'digital' is improperly used: in fact it is considered as digital a vcvs whose output makes discrete voltage transitions at specific time instants or at specific values of the driving signals. The output values have not to be necessarily two (boolean). This type of vcvs are evaluated only once at each time instant of the transient analysis when the Newton algorithm has reached convergence. If the 'digital' vcvs does not change its output value, it is considered that the Newton algorithm has definitely converged; in the other case the vcvs output is updated and the time step is rejected. Please have a look of the <trtime> options for further information.

`trtime` This option has meaning only if the <digital> option is set true (defauls 1ns). Digital devices make only discrete variations of their output voltages. These variations are checked independently from their magnitudes; each time there is a variation between two subsequent integration time instants, this variation is forced to happen in less than <trtime> value. In other words output voltage rise and falling times are ensured to be less than

<trtime>. This parameter helps the simulator to converge, since when there is a transition of a <digital> CCVS the output voltage, in theory, must switch instantaneously. From the simulator point of view this means to shrink the integration time step till the minimum value allowed by the finite precision of the computer arithmetic unit. Obviously this is time consuming since the length of the integration time step governs the simulation speed. To avoid this drawback if the <trtime> is specified, it defines the minimum integration time step, that is <trtime>, below which the transition can be considered satisfing.

`stvar` Integer (0): the number of state variables used in the macromodel

MODEL DEFINITION:

model *ModelName* **vccs** <*parameter=value*> ...

MODEL PARAMETERS:

`macro` Boolean (false): the device is described by an external macromodel.

`module` String (-): the name of the shared module implementing the routines of the macromodel.

`evaluate` String (-): the name of the routine that evaluates the macromodel.

`args` String (-): the name of the routine that handles the arguments passed to the macromodel.

4.3.2 Linear Voltage Controlled Voltage Source

Voltage controlled voltage source is a device with at least 2 terminals; the number of terminals must be even and those above the first 2, is any, are the controlling terminals. There is a <gain> parameter for each controlling pair that linearly relates the output voltage to the voltage across the corresponding input port. External macromodels are supported. The routine describing the macromodel is called with the following ordered arguments

<input> int : an index that uniquely identifies the instance of the VCVS. This index starts from 0 and it is incremented by 1 each time a new instance is found in the netlist; <input> int : number of ports of this VCVS; <input> double *: the array of the port voltages; <input> int : number of state variable; <input> double**: the two vectors of state variable values at the current n time point (index 0) and at the previous (n-1) time point (index 1). In other words, the array can be indexed as 'array[0][StateVarNum]' current time point and 'array[1][StateVarNum]' previous time point. <output> double *: the array of the transconductances (the meaning is the same as that of the <gain> parameter); <output> double *: the value of the branch voltage at the the output port; <input> double : the value of current simulation time. It has meaning only in time domain analyses;

<input> double : the value of simulation time step. It has meaning only in time domain analyses;

The routine handling parameters of the macromodel is called with the arguments

<input> int : an index that uniquely identifies the instance of the VCVS. <input> char * : parameter name; <input> double : parameter value.

ELEMENT DEFINITION:

Name v+ v- <a lot of terminals> **ModelName** <*parameter=value*> ...

ELEMENT PARAMETERS:

`gain1, gain2, ...` Real (0.0): gain of each driving port of the VCVS; since there can be more than one input port, there must be as much of these parameters as the input ports. This applies only if the VCVS is not modelled by an external macromodel. In this case gains are specified by the macromodel.

`gain` Real (0.0): gain of each driving port of the VCVS; since there can be more than one input port, there must be as much of these parameters as the input ports. This applies only if the

VCVS is not modelled by an external macromodel. In this case gains are specified by the macromodel.

`delay` Time delay

`max` Real (-): maximum allowed value of the output voltage.

`min` Real (-): minimum allowed value of the output voltage.

`func` The function that describes the characteristic of the VCVS.

`digital` Boolean (false): tells that this is a digital macromodel. The term 'digital' is improperly used: in fact it is considered as digital a vcvs whose output makes discrete voltage transitions at specific time instants or at specific values of the driving signals. The output values have not to be necessarily two (boolean). This type of vcvs are evaluated only once at each time instant of the transient analysis when the Newton algorithm has reached convergence. If the 'digital' vcvs does not change its output value, it is considered that the Newton algorithm has definitely converged; in the other case the vcvs output is updated and the time step is rejected. Please have a look of the <trtime> options for further information.

`stvar` Integer (0): the number of state variables used in the macromodel

`trtime` Real (1.0e-9): this option has meaning only if the <digital> option is set true. Digital devices make only discrete variations of their output voltages. These variations are checked independently from their magnitudes; each time there is a variation between two subsequent integration time instants, this variation is forced to happen in less than <trtime> value. In other words output voltage rise and falling times are ensured to be less than <trtime>. This parameter helps the simulator to converge, since when there is a transition of a <digital> VCVS the output voltage, in theory, must switch instantaneously. From the simulator point of view this meas to shrink the integration time step till the minimum value allowed by the finite precision of the computer arithmetic unit. Obviously this is time consuming since the length of the integration time step governs the simulation speed and at each transition of a digital vcvs the integration time step is reduced to its allowed minimum. To avoid this drawback, if the <trtime> is specified, it defines the minimum integration time step below which the transition can be considered adequate.

MODEL DEFINITION:

model *ModelName* **vcvs** <*parameter=value*> ...

MODEL PARAMETERS:

`macro` Use device macromodel

`module` Module name containing the macromodel

`evaluate` The name of the macromodel evaluate routine

`args` The name of the routine that handles the arguments passed to the the macromodel

4.3.3 Linear Current Controlled Current Source

Positive current exits the source node and enters the sink node. There are two ways for sensing current: by specifing the sensing nodes or by giving the instance of a sensing device. In the first case a short circuit is placed between the two given nodes. Sense current enters sense+ node and leaves sense- node. In the second case only the source and sink nodes of the current controlled current source are given and a sense device, for example a voltage source, is specified. Its branch current drives the current controlled current source

ELEMENT DEFINITION:

Name sink source <*a lot of terminals*> **cccs** <*parameter=value*> ...

ELEMENT PARAMETERS:

`gain1, ...` Real (0.0): current gain of each driving port of the cccs; since there can be more than one input port, there must be as much of these parameters as the input ports or sensing devices.

`gain` Real (0.0): current gain of each driving port of the cccs; since there can be more than one input port, there must be as much of these parameters as the input ports or sensing devices.

`delay` Time delay

`sensedev` String (-): the name of the instance of the sensing device. The current through this device is used to control the CCCS. If the instance of the CCCS belongs to a subcircuit the sensing device is searched locally. This means that is a local and global devices with the same name exist the local one is chosen. There can be more than one of this parameter according to the number of input ports.

`probe` String (-): the name of the instance of the sensing device. The current through this device is used to control the CCCS. If the instance of the CCCS belongs to a subcircuit the sensing device is searched locally. This means that is a local and global devices with the same name exist the local one is chosen. There can be more than one of this parameter according to the number of input ports.

`gbsensedev` String (-): the global name of the instance of the sensing device. The current through this device is used to control the CCCS. Only global devices are considered; devices at the top-level of the netlist, i.e. those not belonging to subcircuits, are referred to as global. If a global and a local device exist, the global one is selected. If the global one does not exist, an error is displayed. There can be more than one of this parameter according to the number of input ports.

`func` The function that describes the characteristic of the CCCS. The controlling currents can be accessed through the function 'i(px)' where 'x' is an integer number that refers to the input port. For example 'Ic1 o1 o2 i1 i2 cccs func=i(p1)' defines a CCVS with the output port connected between the 'o1' and 'o2' nodes and driven by one input port (a short circuit) connected between the 'i1' and 'i2' nodes. The 'i(p1)' function refers to the first and unique driving port. There is an other mechanism to access driving currents. The circuit element whose current is chosen as driving is identified by the 'sensedev' parameter. In this case in the expression the current can be accessed through the function 'i(v1)' where 'v1' is the name on the instance used as argument in the 'sensedev' parameter; for example in the circuit card 'Ic1 o1 o2 cccs sensedev='v1' func=i(v1)' the current of 'v1' is accessed through the function 'i(v1)'.

`offset` Real (0): the offset current generated by the CCCS. This current is added to that due to the driving signals.

4.3.4 Current Controlled Voltage Source

Positive current exits the source node and enters the sink node; these are the first two nodes of the device. There are two ways for sensing current: by specifing the sensing nodes or by giving the instance of a sensing device. The sensing nodes come always in couple and can be as many as needed. In this case a short circuit is placed between the two given nodes. Sense current enters +sense node and leaves -sense node. Furthermore a mix of sensing nodes and devices can be specified. Each sensing current drives the current controlled voltage source. Gains must be specified if the CCVS is not described by means of an external macromodel. As many as needed sensing devices can be specified, thus the CCVS can be a multi-port device. External macromodels are supported. The routine describing the macromodel is called with the following ordered arguments:

<input> int : an index that uniquely identifies the instance of the CCVS. This index starts from 0 and it is incremented by 1 each time a new instance is found in the netlist; <input> int :

number of ports of this CCVS; <input> double *: the array of the controlling currents; <input> int : number of state variable; <input> double**: the two vectors of state variable values at the current n time point (index 0) and at the previous (n-1) time point (index 1). In other words, the array can be indexed as 'array[0][StateVarNum]' current time point and 'array[1][StateVarNum]' previous time point. <output> double *: the array of the transresistances (the meaning is the same as that of the <gain> parameter); <output> double *: the value of the output port branch voltage; <input> double : the value of simulation time. It has meaning only in time domain analyses.

The voltage of the controlled port can also dependent on the derivative of the currents through the controlling ports. In this case the CCVS acts as a derivator. In time domain analyses, the Gear method of order 1 or 2 is employed to numerically compute derivatives.

ELEMENT DEFINITION:
Name sink source <a lot of terminals> **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:

`gain1, ...` Real (0.0): gain of each driving port of the CCVS; since there can be more than one input port, there must be as much of these parameters as the input ports or sensing devices. This applies only if the CCVS is not modelled by an external macromodel. In this case gains are specified by the macromodel.

`rm` Real (-): transresistance. This parameter is equivalent to <gain1>.

`sensedev` String (-): the name of the instance of the sensing device. The current through this device is used to control the ccvs. If the instance of the CCVS belongs to a subcircuit the sensing device is searched locally. This means that is a local and global devices with the same name exist the local one is chosen. There can be more than one of this parameter according to the number of input ports.

`probe` String (-): the name of the instance of the sensing device. The current through this device is used to control the ccvs. If the instance of the CCVS belongs to a subcircuit the sensing device is searched locally. This means that is a local and global devices with the same name exist the local one is chosen. There can be more than one of this parameter according to the number of input ports.

`gbsensedev` String (-): the global name of the instance of the sensing device. The current through this device is used to control the ccvs. Only global devices are considered; devices at the top-level of the netlist, i.e. those not belonging to subcircuits, are referred to as global. If a global and a local device exist, the global one is selected. If the global one does not exist, an error is displayed. There can be more than one of this parameter according to the number of input ports.

`func` The function that describes the characteristic of the CCVS. The controlling currents can be accessed through the function 'i(px)' where 'x' is an integer number that refers to the input port. For example 'Ic1 o1 o2 i1 i2 cccs func=i(p1)' defines a CCVS with the output port connected between the 'o1' and 'o2' nodes and driven by one input port (a short circuit) connected between the 'i1' and 'i2' nodes. The 'i(p1)' function refers to the first and unique driving port. There is an other mechanism to access driving currents. The circuit element whose current is chosen as driving is identified by the 'sensedev' parameter. In this case in the expression the current can be accessed through the function 'i(v1)' where 'v1' is the name on the instance used as argument in the 'sensedev' parameter; for example in the circuit card 'Ic1 o1 o2 cccs sensedev='v1' func=i(v1)' the current of 'v1' is accessed through the function 'i(v1)'.

`digital` It is a digital macromodel. It is supposed that the output varied sharply between two

different and constant voltage levels.

`trtime` This option has meaning only if the <digital> option is set true (defauls 1ns). Digital devices make only discrete variations of their output voltages. These variations are checkes independently from their magnitudes; each time there is a variation between two subsequent integration time instants, this variation is forced to happen in less than <trtime> value. In other words output voltage rise and falling times are ensured to be less than <trtime>. This parameter helps the simulator to converge, since when there is a transition of a <digital> CCVS the output voltage, in theory, must switch instantaneously. From the simulator point of view this meas to shrink the integration time step till the minimum value allowed by the finite precision of the computer arithmetic unit. Obviously this is time consuming since the length of the integratio time step governs the simulation speed. To avoid this drawback if the <trtime> is specified; it defined the minimum integration time step, that is <trtime>, below which the transition can be considered satisfing.

`der1, ...` Real (0.0): the CCVS can act as a differentiator: the branch voltage of the controlled port (port 1) can be equal to the derivatives of the current through the controlling ports (port 2, port3, ...) multiplied by the <der> values. Since there can be more than one controlling port, this parameter can be specified for each controlling port. Note that derivatives are considered as additive with respect to the input/output gains. These parameters apply only if the CCVS is not modelled by an external macromodel; in this case derivative values are specified in the macromodel. The numerical method employed to implement the differentiator is the Gear one of order 1 and 2.

`delay` Time delay

`stvar` Number of state variable used in the macromodel

MODEL DEFINITION:

model *ModelName* **ccvs** <*parameter=value*> ...

MODEL PARAMETERS:

`macro` Use device macromodel

`module` Shared module containing the macromodel

`evaluate` The name of the macromodel evaluating routine

4.3.5 Linear Two Winding Ideal Transformer

This element implements an ideal transformer, which implies that it works also at DC. Winding 1 connects terminals w1+ and w1-, and similarly winding 2 connects w2+ and w2-. The number of turns on windings 1 and 2 are given respectively by t1 and t2, they must not be zero. The absolute number of turns of each winding is not important, only the ratio. Both t1 and t2 are modifiable, 1 is the default. Current through winding 1 is computed. Note that port 1 is described by using the normal convention for its voltage and current, while port 2 is described with the non-normal convention, so that the v2*i2 product corresponds to the power flowing out from the port 2 (i.e. v1*i1 + v2*i2 = 0).

To model a physical transformer with L1 and L2 being the inductance of the windings and K being the coupling coefficient, add an inductor Lm=K*L1 in parallel with winding 1 and inductors Le1=L1*(1-K) and Le2=L2*(1-K) in series with windings 1 and 2 respectively. The turns ratio can be computed with t1/t2 = sqrt(L1/L2). K can be calculated from the L1 (the inductance of winding 1 with winding 2 open) and Ls (the inductance of winding 1 with winding 2 shorted) with K=sqrt(1-Ls/L1).

t1:t2 o o o—Le1—o w1+ o—-) ||| (—-o w2+ o—Le2—o |) ||| (Lm) ||| (|) ||| (o w1- o—-) ||| (—-o w2-

ELEMENT DEFINITION:
Name p1+ p1- s2+ s2- **transformer** *<parameter=value> ...*

ELEMENT PARAMETERS:
`t1` Number of turns on winding 1
`t2` Number of turns on winding 2

4.3.6 Ideal Nullor

Nullor is a two-port device; port 1 is identified by the first two nodes of the device and port 2 by the remaining two nodes. Its characteristic at port 1 is described by the i1 branch current and v1 branch voltage. Both are always null: i1 = 0 and v1 = 0. The characteristic of port 1 is composed of only a point located in the origin of the voltage/current axes (nullator). Port 2 is described by current i2 and voltage v2; they are not related and can independently assume any value (norator).

ELEMENT DEFINITION:
Name p1+ p1- s2+ s2- **nullor**

4.3.7 Coupled Inductors

This instance is not a "true" one, but it allows coupling of two or more inductors. It must be specified the name of at least two inductor instances through a parameter and the corresponding coupling coefficient. If an inductor is coupled to more than one other inductor a sequence of inductance instances must be specified with the corresponding coupling coefficient. The first inductance of this sequence is taken as the reference inductor, the other ones are those to which the reference inductor is coupled to. Note that coupling is mutual.

ELEMENT DEFINITION:
Name **coupledind** *<parameter=value> ...*

ELEMENT PARAMETERS:
`ind1, ind2, ...` This parameter specifies the instance of another inductor to which there is an inductive coupling. There must be two ore more of these parameters since coupling can take place only between two or more inductors. The first specified inductor is taken as reference. Note that the other specified inductor is automatically (mutually) coupled to the reference one.
`k1, k2, ...` This parameter specifies the coupling coefficient between two inductors. Its value must be in the range [-1,+1].

4.3.8 Polynomial Transconductor

The poly element is a 4 terminal nonlinear resistive element whose output current is given by

$$i = (n0 + n1 * v + n2 * v^2 + ... + n8 * v^8)/(d0 + d1 * v + d2 * v^2 + ... + d8 * v^8)$$

where 'v' is the voltage between the sense nodes in volt and 'i' is the output current in ampere. Positive current exits the source node and enters the sink node. Noise is considered only if the element behaves like a nonlinear polinomial resistor, that is, the sink/source nodes coincide with the sense+/sense- ones.

ELEMENT DEFINITION:
Name sink source sense+ sense- **ModelName** <*parameter=value*> ...

ELEMENT PARAMETERS:
`n0` Real (0.0): constant term at numerator.
`n1` Real (0.0): coefficient of first degree term of the numerator.
`n2` Real (0.0): coefficient of second degree term at numerator.
`n3` Real (0.0): coefficient of third degree term at numerator.
`n4` Real (0.0): coefficient of fourth degree term at numerator.
`n5` Real (0.0): coefficient of fifth degree term at numerator.
`n6` Real (0.0): coefficient of sixth degree term at numerator.
`n7` Real (0.0): coefficient of seventh degree term at numerator.
`n8` Real (0.0): coefficient of eighth degree term at numerator.
`d0` Real (1.0): constant term at denominator.
`d1` Coefficient of first degree term at denominator.
`d2` Coefficient of second degree term at denominator.
`d3` Coefficient of third degree term at denominator.
`d4` Coefficient of fourth degree term at denominator.
`d5` Coefficient of fifth degree term at denominator.
`d6` Coefficient of sixth degree term at denominator.
`d7` Coefficient of seventh degree term at denominator.
`d8` Coefficient of eighth degree term at denominator.
`tnom` Real (27): nominal temperature at which the polynomial characteristic was determined.
`tc1` Real (0.0): coefficient of the linear term in the temperature dependence relation.
`tc2` Real (0.0): coefficient of the quadratic term in the temperature dependence relation.
`noisetype` Integer (5): types of generated noise; possible choices are:
0x0 : noiseless element; 0x1 : thermal noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.
MODEL DEFINITION:
model *ModelName* **poly** <*parameter=value*> ...

MODEL PARAMETERS:
`af` Real (1.0): flicker noise exponent.
`kf` Real (0.0): flicker noise coefficient.

4.4 semiconductor device

4.4.1 Junction Diode

The diode is tied between two pins: ‘anode’ and ‘cathode’. The other three optional pins: ‘thermal’, ‘didv’ and ‘didT’ are internally connected to independent current sources. The ‘thermal’ pin refers to the heat-flow from the diode and is active only during an eletro-thermal analysis. It must be connected to a node of the thermal network. Heat flow is equal to the electrical power through the electrical port and is modeled by a current source tied between ground and the ‘thermal’ node; the related current exits from the ‘thermal’ node.The ‘didT’ pin refers to the derivative of the diode branch current with respect to diode branch voltage. Finally the ‘didT’ pin refers to the derivative of the diode branch current with respect to juntion temperature. These optional pins may be made inactive if connected to ground

ELEMENT DEFINITION:

Name anode cathode [thermal] [didv] [didT] **ModelName** <*parameter=value*> ...

ELEMENT PARAMETERS:

`area` Junction area in square meters
`perimeter` Junction perimeter in meters
`peri` Junction perimeter in meters
`perim` Junction perimeter in meters
`pj` Junction perimeter in meters
`region` DC operating region, 0=off, 1=on
`temp` Device junction themperature
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless diode; 0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.
`m` Integer (1): number of devices connected in parallel.

MODEL DEFINITION:

model *ModelName* **diode** <*parameter=value*> ...

MODEL PARAMETERS:

`is` Saturation current
`js` Saturation current
`jsw` Real (0.0): sidewall saturation current.
`isw` Real (0.0): sidewall saturation current.

`area` Junction area in square meters
`rs` Real (0.0): parassitic resistor connected in series to the diode that accounts teh resistance of contacts and of silicon that does not constitute the junction.
`rsw` Real (0.0): sidewall series resistance.
`n` Emission coefficient
`ik` Real (-): forward injection knee current.
`ikf` Real (-): forward injection knee current.
`ikp` Real (-): side wall forward injection knee current.
`tt` Real (0.0): transit time.
`cjo` Real (0.0): zero-bias junction capacitance
`cj0` Real (0.0): zero-bias junction capacitance
`vj` Junction potential
`pb` Junction potential
`m` Grading coefficient
`mj` Grading coefficient
`cjsw` Real (0.0): zero-bias sidewall junction capacitance
`cjp` Real (0.0): zero-bias sidewall junction capacitance
`vjsw` Sidewall Junction potential
`mjsw` Sidewall Grading coefficient
`eg` Real (1.11): band gap.
`xti` Real (3.0): saturation current temperature exponent.
`pt` Real (3.0): saturation current temperature exponent.
`fc` Forward-bias depletion capacitance threshold
`rl` Real (1.0e12): leakage resistance connected in parallel to the diode. Note that the default 'gmin' conductance is also connected in parallel
`bv` Reverse breakdown voltage

`ibv` Real (0.001): current at breakdown voltage.
`imax` Real (1.0): the maximum current through the diode above which its electrical characteristic is linearized. It is also assumed that above this current the diode is melting.

`tnom` Parameter measurement temperature
`trs1` Real (0.0;): series resistance first order temperature coefficient.
`trs` Real (0.0;): series resistance first order temperature coefficient.
`trs2` Real (0.0;): series resistance second order temperature coefficient.

`kf` Real (0.0): flicker noise coefficient.
`af` Real (1.0): flicker noise exponent.

`tlev` DC temperature selector.
`tlevc` AC temperature selector.
`tvj` Temperature coefficient of vj parameter.
`tvjsw` Temperature coefficient of vjsw parameter.
`tcjo` Temperature coefficient of cjo parameter.
`tcjsw` Temperature coefficient of cjsw parameter.
`pta` Junction potential temperature coefficient.
`ptp` Sidewall junction potential temperature coefficient.
`cta` Junction capacitance temperature coefficient.
`ctp` Sidewall junction capacitance temperature coefficient.
`level` Model selector
`ns` Sidewall emission coefficient

4.4.2 Bipolar Junction Transistor

The bipolar transistor model is adapted from the integral charge model of Gummel and Poon, and includes several high bias level effects. The model also includes a collector-substrate capacitor that is not installed if the collector and substrate terminals are connected to the same node. It defaults to the simpler Ebers-Moll model if certain parameters are left unspecified

ELEMENT DEFINITION:
Name collector base emitter [substrate] [thermal] **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:
`area` Junction area in square meters
`region` DC operating region, 0=off, 1=on, 2=rev, 3=sat
`temp` Device junction temperature
`m` Integer (1): number of devices connected in parallel.
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless BJT; 0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.
MODEL DEFINITION:
model *ModelName* **bjt** *<parameter=value> ...*

MODEL PARAMETERS:
`npn` NPN bipolar transistor
`pnp` PNP bipolar transistor
`type` String (- : type of bipolar transistor; allowed values are 'npn' or 'pnp'.

`is` Saturation current
`js` Saturation current
`bf` Forward beta
`nf` Forward emission coefficient
`vaf` Forward early voltage
`vbf` Forward early voltage
`va` Forward early voltage
`ikf` High current corner for forward beta
`jbf` High current corner for forward beta
`ik` High current corner for forward beta
`ise` B-E leakage saturation current
`jle` B-E leakage saturation current
`ne` B-E leakage emission coef
`nle` B-E leakage emission coef
`br` Reverse beta
`nr` Reverse emission coefficient
`var` Reverse early voltage
`vbr` Reverse early voltage
`ikr` High current corner for reverse beta
`jbr` High current corner for reverse beta
`isc` B-C leakage saturation current
`jlc` B-C leakage saturation current
`nc` B-C leakage emission coef
`nlc` B-C leakage emission coef
`rb` Zero bias base resistance (default 1.0e-3)
`irb` Current for base resistance midpoint
`jrb` Current for base resistance midpoint
`rbm` Minimum base resistance for high currents
`re` Emitter resistance (default 1.0e-3)
`rc` Collector resistance (default 1.0e-3)
`imax` The currents through the BE and BC junctions exponentially depend on the Vbe and Vbc voltages. The 'imax' parameter defines the maximum Vbe and Vbc voltages above which the exponential function is linearized. Linearization is introduced to avoid numerical overflow. The value of this parameter can be chosen close to the maximum allowed currents of the BJT. The default value of this parameter is 1A.
`cje` Real (0.0): base-emitter zero-bias junction capacitance
`vje` B-E built-in junction potential
`pe` B-E built-in junction potential
`mje` B-E junction exponent
`me` B-E junction exponent
`cjc` B-C zero-bias junction capacitance
`vjc` B-C built-in junction potential
`pc` B-C built-in junction potential
`mjc` B-C junction exponent
`mc` B-C junction exponent
`xcjc` Fraction of B-C capacitance tied to internal base node
`cdis` Fraction of B-C capacitance tied to internal base node
`cjs` Real (0.0): c-substrate zero-bias junction capacitance
`ccs` Real (0.0): c-substrate zero-bias junction capacitance

`vjs` C-Substrate built-in junction potential
`mjs` Substrate junction exponent
`ms` Substrate junction exponent
`fc` Junction capacitor forward-bias threshold
`tf` Ideal forward transit time
`xtf` Coefficient for bias dependence of 'tf'
`vtf` Voltage describing Vbc dependence of 'tf'
`itf` High current parameter for effect on 'tf'
`jtf` High current parameter for effect on 'tf'
`ptf` Excess phase at freq = 1.0/(tf*2pi) Hz
`tr` Real (0.0: ideal reverse transit time.
`xtb` Beta temperature exponent
`tb` Beta temperature exponent
`eg` Band gap
`xti` Temperature exponent for effect on Js
`pt` Temperature exponent for effect on Js
`tnom` Parameter measurement temperature

`kf` Real (0.0): flicker noise coefficient.
`af` Real (1.0): flicker noise exponent.

4.4.3 Junction Field Effect Transistor

The JFET model is derived from the FET model of Shichman and Hodges. JFETs require the use of a model statement.

ELEMENT DEFINITION:
Name drain gate source **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`area` Junction area in square meters
`region` DC operating region 0=off 1=on 2=rev 3=ohmic
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless JFET; 0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.
MODEL DEFINITION:
model *ModelName* **jfet** <*parameter*=*value*> ...

MODEL PARAMETERS:
`nfet` N type JFET
`njf` N type JFET
`pfet` P type JFET
`pjf` P type JFET
`vto` Pinch-off voltage
`beta` Transconductance parameter
`lambda` Channel length modulation parameter
`rd` Drain resistance
`rs` Source resistance
`is` Gate saturation current
`js` Gate saturation current

`cgs` Zero bias gate-source junction capacitance
`cgd` Zero bias gate-drain junction capacitance
`pb` Gate junction potential
`fc` Forward-bias depletion capacitance threshold
`tnom` Parameter measurement temperature
`kf` Flicker-noise coefficient
`af` Flicker-noise exponent
`imax` Explosion current

4.4.4 Philips M903 mosfet transistor

The m903 mosfet transistor developes by Philips.

ELEMENT DEFINITION:
Name drain gate source bulk **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`w` W
`l` L
`mult` Integer (1): number of identical mosfets connected in parallel.
`m` Integer (1): number of identical mosfets connected in parallel.
`as` Real (-): source area.
`ad` Real (-): drain area.
`ps` Real (-): source perimeter.
`pd` Real (-): drain perimeter.
MODEL DEFINITION:
model *ModelName* **m903** <*parameter*=*value*> ...

MODEL PARAMETERS:
`nmos` Nmos
`pmos` Pmos
`ler` LER
`wer` WER
`lvar` LVAR
`lap` LAP
`wvar` WVAR
`wot` WOT
`tr` TR
`vtor` VTOR
`stvto` STVTO
`slvto` SLVTO
`sl2vto` SL2VTO
`sl3vto` SL3VTO
`swvto` SWVTO
`kor` KOR
`slko` SLKO
`sl2ko` SL2KO
`swko` SWKO
`kr` KR
`slk` SLK

`sl2k` SL2K
`swk` SWK
`phibr` PHIBR
`vsbxr` VSBXR
`slvsbx` SLVSBX
`swvsbx` SWVSBX
`betsq` BETSQ
`etabet` ETABET
`lp1` LP1
`fbet1` FBET1
`lp2` LP2
`fbet2` FBET2
`the1r` THE1R
`stthe1r` STTHE1R
`slthe1r` SLTHE1R
`stlthe1` STLTHE1
`gthe1` GTHE1
`swthe1` SWTHE1
`wdog` WDOG
`fthe1` FTHE1
`the2r` THE2R
`stthe2r` STTHE2R
`slthe2r` SLTHE2R
`stlthe2` STLTHE2
`swthe2` SWTHE2
`the3r` THE3R
`stthe3r` STTHE3R
`slthe3r` SLTHE3R
`stlthe3` STLTHE3
`swthe3` SWTHE3
`gam1r` GAM1R
`slgam1` SLGAM1
`swgam1` SWGAM1
`etadsr` ETADSR
`alpr` ALPR
`etaalp` ETAALP
`slalp` SLALP
`swalp` SWALP
`vpr` VPR
`gamoor` GAMOOR
`slgamoo` SLGAMOO
`sl2gamoo` SL2GAMOO
`etagamr` ETAGAMR
`mor` MOR
`stmo` STMO
`slmo` SLMO
`etamr` ETAMR
`zet1r` ZET1R
`etazet` ETAZET

`slzet1` SLZET1
`vsbtr` VSBTR
`slvsbt` SLVSBT
`a1r` A1R
`sta1` STA1
`sla1` SLA1
`swa1` SWA1
`a2r` A2R
`sla2` SLA2
`swa2` SWA2
`a3r` A3R
`sla3` SLA3
`swa3` SWA3
`tox` TOX
`col` COL
`ntr` NTR
`nfmod` NFMOD
`nfr` NFR
`nfar` NFAR
`nfbr` NFBR
`nfcr` NFCR
`l` L
`w` W
`dta` DTA
`th3mod` TH3MOD

4.4.5 MOSFET level 2 device

The mosfet transistor model is the two one of Berkeley SPICE3.

ELEMENT DEFINITION:
Name drain gate source bulk **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`l` Real (2.0e-6): channel length.
`w` Real (2.0e-6): channel width.
`ad` Drain region area.
`as` Source region area.
`pd` Drain perimeter.
`ps` Source perimeter.
`nrd` Drain diffusion sheet resistance.
`nrs` Source diffusion sheet resistance.
`region` Operation region; it is used to compute the guess to star the Newton method during the computation of the DC operating point. Allowed values are region=0 (off), region=1 (on); in the first case the Vgs voltage is set equal to Vth and Vds = 2 * Vth; in the second case Vgs = 0. Default value of parameter is region=1.
`temp` Device junction temperature.
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless resistor; 0x1 : thermal noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

MODEL DEFINITION:

model *ModelName* **mos2** <*parameter=value*> ...

MODEL PARAMETERS:

`nmos` Nmos type transistor
`pmos` Pmos type transistor
`type` String (-): the mosfet type. Available keywords are <n> and <p>

`cbd` Real (0.0): bulk-drain junction capacitance.
`cbs` Real (0.0): bulk-drain junction capacitance.
`gamma` Bulk threshold parameter
`kp` Transconductance parameter
`phi` Surface potential
`is` Bulk junction saturation current
`pb` Bulk junction potential
`cgso` Real (0.0): gate-source overlap cap.
`cgdo` Real (0.0): gate-drain overlap cap.
`cgbo` Real (0.0): gate-bulk overlap cap.
`cj` Bottom junction capacitance per area
`mj` Bottom grading coefficient
`cjsw` Side junction capacitance per area
`mjsw` Side grading coefficient
`neff` Total channel charge coefficient
`lambda` Channel length modulation
`l` Default length of the mos model
`is` Bulk junction saturation current
`js` Bulk juncction saturation current density
`tox` Real (1.0e-7): oxide thickness.
`ld` Lateral diffusion
`uo` Surface mobility
`fc` Forward bias juction fit parm.
`nsub` Substrate doping
`tpg` Gate type
`nss` Surface state density
`delta` Width effect on threshold
`vmax` Maximum carrier drift velocity
`xj` Metallurgical junction depth
`nfs` Fast surface state density
`delta` Width effect on threshold
`tnom` Parameter measurement temperature (Celsius)
`vt0` Threshold voltage
`vto` Threshold voltage
`w` Default width of the mos model
`uexp` Critical field exponent in mobility degradation
`ucrit` Critial field for mobility degradation

`rsh` Sheet resistance
`rd` Drain ohmic resistance (default 10 mOhm)
`rs` Source ohmic resistance (default 10 mOhm)

`imax` Real (100.0e-3): when current is above this value, the diode is considered in the 'melting region' and its characteristic is linearized.
`vbox` Real (1.0e9*tox): oxide breakdown voltage.

`kf` Real (0.0): flicker noise coefficient.
`af` Real (1.0): flicker noise exponent.

4.4.6 MOSFET level 3 device

The mosfet transistor model is the level three of Berkeley SPICE3.

ELEMENT DEFINITION:
Name drain gate source bulk [thermal] **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`l` Real (2.0e-6): channel length.
`w` Real (2.0e-6): channel width.
`ad` Drain region area.
`as` Source region area.
`pd` Drain perimeter.
`ps` Source perimeter.
`nrd` Drain diffusion sheet resistance.
`nrs` Source diffusion sheet resistance.
`region` Operation region; it is used to compute the guess to star the Newton method during the computation of the DC operating point. Allowed values are region=0 (off), region=1 (on); in the first case the Vgs voltage is set equal to Vth and Vds = 2 * Vth; in the second case Vgs = 0. Default value of parameter is region=1.
`temp` Device junction temperature.
MODEL DEFINITION:
model *ModelName* **mos3** <*parameter*=*value*> ...

MODEL PARAMETERS:
`cbd` B-D junction capacitance
`cbs` B-S junction capacitance
`gamma` Bulk threshold parameter
`kp` Transconductance parameter
`nmos` Nmos type transistor.
`phi` Surface potential
`pmos` Pmos type transistor.
`imax` Real (100.0e-3): when current is above this value, the diode is considered in the 'melting region' and its characteristic is linearized.
`is` Bulk junction saturation current
`pb` Bulk junction potential
`cgso` Gate-source overlap cap.
`cgdo` Gate-drain overlap cap.
`cgbo` Gate-bulk overlap cap.
`cj` Bottom junction capacitance per area
`mj` Bottom grading coefficient
`cjsw` Side junction capacitance per area

`mjsw` Side grading coefficient
`l` Default length of the mos model.
`is` Bulk junction saturation current
`js` Bulk juncction saturation current density
`tox` Oxide thickness
`ld` Lateral diffusion
`uo` Surface mobility
`fc` Forward bias juction fit parm.
`nsub` Substrate doping
`tpg` Gate type
`nss` Surface state density
`delta` Width effect on threshold
`vmax` Maximum carrier drift velocity
`xj` Junction depth
`nfs` Fast surface state density
`eta` Drain surce voltage effect on the threshold voltage
`xd` Depletion layer width
`delta` Width effect on threshold
`theta` Mobility modulation
`alpha` Alpha
`kappa` Kappa
`tnom` Parameter measurement temperature (Celsius).
`vt0` Threshold voltage
`vto` Threshold voltage
`w` Default width of the mos model.

`rsh` Sheet resistance
`rds` Real (infinite): the resistance connected between drain and source terminal (internal ones) of the mosfet.
`rd` Drain ohmic resistance (default 10 mOhm)
`rs` Source ohmic resistance (default 10 mOhm)

4.4.7 Berkeley Short Channel IGFET Model Version-3

The bsim3 mosfet transistor model developed at the Berkeley University. Clusters of this model are supported. A cluster is a list of models; when an element instance refers to a cluster of models, the list is searched. The <wmin> <wmax> <lmin> and <lmax> values speficied in each model of the cluster are compared to the current values of the <w> and <l> parameters of the mosfet instance. Then <w> and <l> fall inbetween the above minimum and maximum values the corresponding model is picked up from the cluster and bound to the mosfet instance.

ELEMENT DEFINITION:
Name drain gate source bulk **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:
`w` Channel width
`l` Channel length
`ad` Drain region area
`as` Source region area
`pd` Drain perimeter

`ps` Source perimeter
`nrd` Drain diffusion sheet resistance
`nrs` Source diffusion sheet resistance
`temp` Device junction temperature
`m` Integer (1): number of devices connected in parallel.
`sa` Distance between OD edge to poly of one side.
`sb` Distance between OD edge to poly of one other side.
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless BSIM3; 0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

MODEL DEFINITION:

model *ModelName* **bsim3** <*parameter*=*value*> ...

MODEL PARAMETERS:

`ijth` Real (1.0e-1): diode maximum current above which it is considered as melting and its characteristic is linearised to aid numerical methods to converge.
`imax` Real (1.0e-1): diode maximum current above which it is considered as melting and its characteristic is linearised to aid numerical methods to converge.
`vbox` Real (1.0e9*tox): oxide breakdown voltage.

`capmod` Capacitance model selector. It is an integer number between 0 and 3 (default 3).
`mobmod` Mobility model selector
`noimod` Integer (1): noise model selector. Possible values are:
1: MOS2 model adopted for channel thermal noise and fliker noise; 2: BSIM3 model adopted for channel thermal noise and fliker noise; 3: MOS2 model adopted for channel thermal noise and BSIM3 for fliker noise; 4: BSIM3 model adoted for channel thermal noise and MOS2 for fliker noise.
`quiet` Boolean (false): when true, warning messages about parameters whose values are out of the allowed range are given.
`binunit` Bin unit selector
`tox` Gate oxide thickness in meters
`nqsmod` Non-quasi-static model selector. It is a boolean flag (default false).
`nmos` Nmos type of mosfet
`toxm` Gate oxide thickness used in extraction
`cdsc` Drain/Source and channel coupling capacitance
`cdscb` Body-bias dependence of cdsc
`cdscd` Drain-bias dependence of cdsc
`cit` Interface state capacitance
`nfactor` Subthreshold swing Coefficient
`xj` Junction depth in meters
`vsat` Saturation velocity at tnom
`at` Temperature coefficient of vsat
`a0` Non-uniform depletion width effect coefficient.
`ags` Gate bias coefficient of Abulk.
`a1` Non-saturation effect coefficient
`a2` Non-saturation effect coefficient
`keta` Body-bias coefficient of non-uniform depletion width effect.
`nsub` Substrate doping concentration
`nch` Channel doping concentration

`ngate` Poly-gate doping concentration
`gamma1` Vth body coefficient
`gamma2` Vth body coefficient
`vbx` Vth transition body Voltage
`vbm` Maximum body voltage
`pmos` Pmos type of mosfet
`xt` Doping depth
`k1` Bulk effect coefficient 1
`kt1` Temperature coefficient of Vth
`kt1l` Temperature coefficient of Vth
`kt2` Body-coefficient of kt1
`k2` Bulk effect coefficient 2
`k3` Narrow width effect coefficient
`k3b` Body effect coefficient of k3
`w0` Narrow width effect parameter
`nlx` Lateral non-uniform doping effect
`dvt0` Short channel effect coeff. 0
`dvt1` Short channel effect coeff. 1
`dvt2` Short channel effect coeff. 2
`dvt0w` Narrow Width coeff. 0
`dvt1w` Narrow Width effect coeff. 1
`dvt2w` Narrow Width effect coeff. 2
`drout` DIBL coefficient of output resistance
`dsub` DIBL coefficient in the subthreshold region
`vth0` Threshold voltage
`vtho` Threshold voltage
`ua` Linear gate dependence of mobility
`ua1` Temperature coefficient of ua
`ub` Quadratic gate dependence of mobility
`ub1` Temperature coefficient of ub
`uc` Body-bias dependence of mobility
`uc1` Temperature coefficient of uc
`u0` Low-field mobility at Tnom
`ute` Temperature coefficient of mobility
`voff` Threshold voltage offset
`tnom` Parameter measurement temperature
`cgso` Gate-source overlap capacitance per width
`cgdo` Gate-drain overlap capacitance per width
`cgbo` Gate-bulk overlap capacitance per length
`xpart` Channel charge partitioning
`elm` Non-quasi-static Elmore Constant Parameter
`delta` Effective Vds parameter
`rsh` Source-drain sheet resistance
`rdsw` Source-drain resistance per width
`prwg` Gate-bias effect on parasitic resistance
`prwb` Body-effect on parasitic resistance
`prt` Temperature coefficient of parasitic resistance
`eta0` Subthreshold region DIBL coefficient
`etab` Subthreshold region DIBL coefficient

`pclm` Channel length modulation Coefficient
`pdiblc1` Drain-induced barrier lowering coefficient
`pdiblc2` Drain-induced barrier lowering coefficient
`pdiblcb` Body-effect on drain-induced barrier lowering
`pscbe1` Substrate current body-effect coefficient
`pscbe2` Substrate current body-effect coefficient
`pvag` Gate dependence of output resistance parameter
`js` Source/drain junction reverse saturation current density
`jsw` Sidewall junction reverse saturation current density
`pb` Source/drain junction built-in potential
`nj` Source/drain junction emission coefficient
`n` Source/drain junction emission coefficient
`xti` Junction current temperature exponent
`mj` Source/drain bottom junction capacitance grading coefficient
`pbsw` Source/drain sidewall junction capacitance built in potential
`mjsw` Source/drain sidewall junction capacitance grading coefficient
`pbswg` Source/drain (gate side sidewall junction capacitance built in potential
`mjswg` Source/drain (gate side sidewall junction capacitance grading coefficient
`cj` Source/drain bottom junction capacitance per unit area
`vfbcv` Flat Band Voltage parameter for capmod=0 only
`vfb` Flat Band Voltage
`cjsw` Source/drain sidewall junction capacitance per unit periphery
`cjswg` Source/drain (gate side sidewall junction capacitance per unit width
`tpb` Temperature coefficient of pb
`tcj` Temperature coefficient of cj
`tpbsw` Temperature coefficient of pbsw
`tcjsw` Temperature coefficient of cjsw
`tpbswg` Temperature coefficient of pbswg
`tcjswg` Temperature coefficient of cjswg
`acde` Exponential coefficient for finite charge thickness
`moin` Coefficient for gate-bias dependent surface potential
`noff` C-V turn-on/off parameter
`voffcv` C-V lateral-shift parameter
`lint` Length reduction parameter
`ll` Length reduction parameter
`llc` Length reduction parameter for CV
`lln` Length reduction parameter
`lw` Length reduction parameter
`lwc` Length reduction parameter for CV
`lwn` Length reduction parameter
`lwl` Length reduction parameter
`lwlc` Length reduction parameter for CV
`lmin` Real (0.0): Minimum length for the model used as a selection criterion when the model belongs to a cluster of models. When the model does not belongs to a cluster and <l> is less than <lmin> an error is given. The error can be a fatal one, that cause the abortion of the simulation or a simply a warning one. The severity of this error can be chosen through the <bounderror> parameter.
`lmax` Real (1.0): maximum length for the model used as a selection criterion when the model belongs to a cluster of models. When the model does not belongs to a cluster and <l>

is greather than <lmax> an error is given. The error can be a fatal one, that cause the abortion of the simulation or a simply a warning one. The severity of this error can be chosen through the <bounderror> parameter.

`wr` Width dependence of rds

`wint` Width reduction parameter

`dwg` Width reduction parameter

`dwb` Width reduction parameter

`wl` Width reduction parameter

`wlc` Width reduction parameter for CV

`wln` Width reduction parameter

`ww` Width reduction parameter

`wwc` Width reduction parameter for CV

`wwn` Width reduction parameter

`wwl` Width reduction parameter

`wwlc` Width reduction parameter for CV

`wmin` Real (0.0): minimum width for the model used as a selection criterion when the model belongs to a cluster of models. When the model does not belongs to a cluster and <w> is less than <wmin> an error is given. The error can be a fatal one, that cause the abortion of the simulation or a simply a warning one. The severity of this error can be chosen through the <bounderror> parameter.

`wmax` Real (1.0): maximum width for the model used as a selection criterion when the model belongs to a cluster of models. When the model does not belongs to a cluster and <w> is less than <wmax> an error is given. The error can be a fatal one, that cause the abortion of the simulation or a simply a warning one. The severity of this error can be chosen through the <bounderror> parameter.

`b0` Abulk narrow width parameter

`b1` Abulk narrow width parameter

`cgsl` New C-V model parameter

`cgdl` New C-V model parameter

`ckappa` New C-V model parameter

`cf` Fringe capacitance parameter

`clc` Vdsat parameter for C-V model

`cle` Vdsat parameter for C-V model

`dwc` Delta W for C-V model

`dlc` Delta L for C-V model

`alpha0` substrate current model parameter

`alpha1` substrate current model parameter

`beta0` substrate current model parameter

`lcdsc` Length dependence of cdsc

`lcdscb` Length dependence of cdscb

`lcdscd` Length dependence of cdscd

`lcit` Length dependence of cit

`lnfactor` Length dependence of nfactor

`lxj` Length dependence of xj

`lvsat` Length dependence of vsat

`lat` Length dependence of at

`la0` Length dependence of a0

`lags` Length dependence of ags

`la1` Length dependence of a1

`la2` Length dependence of a2
`lketa` Length dependence of keta
`lnsub` Length dependence of nsub
`lnch` Length dependence of nch
`lngate` Length dependence of ngate
`lgamma1` Length dependence of gamma1
`lgamma2` Length dependence of gamma2
`lvbx` Length dependence of vbx
`lvbm` Length dependence of vbm
`lxt` Length dependence of xt
`lk1` Length dependence of k1
`lkt1` Length dependence of kt1
`lkt1l` Length dependence of kt1l
`lkt2` Length dependence of kt2
`lk2` Length dependence of k2
`lk3` Length dependence of k3
`lk3b` Length dependence of k3b
`lw0` Length dependence of w0
`lnlx` Length dependence of nlx
`ldvt0` Length dependence of dvt0
`ldvt1` Length dependence of dvt1
`ldvt2` Length dependence of dvt2
`ldvt0w` Length dependence of dvt0w
`ldvt1w` Length dependence of dvt1w
`ldvt2w` Length dependence of dvt2w
`ldrout` Length dependence of drout
`ldsub` Length dependence of dsub
`lvtho` Length dependence of vto
`lvth0` Length dependence of vto
`lua` Length dependence of ua
`lua1` Length dependence of ua1
`lub` Length dependence of ub
`lub1` Length dependence of ub1
`luc` Length dependence of uc
`luc1` Length dependence of uc1
`lu0` Length dependence of u0
`lute` Length dependence of ute
`lvoff` Length dependence of voff
`lelm` Length dependence of elm
`ldelta` Length dependence of delta
`lrdsw` Length dependence of rdsw
`lprwg` Length dependence of prwg
`lprwb` Length dependence of prwb
`lprt` Length dependence of prt
`leta0` Length dependence of eta0
`letab` Length dependence of etab
`lpclm` Length dependence of pclm
`lpdiblc1` Length dependence of pdiblc1
`lpdiblc2` Length dependence of pdiblc2

`lpdiblcb` Length dependence of pdiblcb
`lpscbe1` Length dependence of pscbe1
`lpscbe2` Length dependence of pscbe2
`lpvag` Length dependence of pvag
`lwr` Length dependence of wr
`ldwg` Length dependence of dwg
`ldwb` Length dependence of dwb
`lb0` Length dependence of b0
`lb1` Length dependence of b1
`lcgsl` Length dependence of cgsl
`lcgdl` Length dependence of cgdl
`lckappa` Length dependence of ckappa
`lcf` Length dependence of cf
`lclc` Length dependence of clc
`lcle` Length dependence of cle
`lalpha0` Length dependence of alpha0
`lalpha1` Length dependence of alpha1
`lbeta0` Length dependence of beta0
`lvfbcv` Length dependence of vfbcv
`lvfb` Length dependence of vfb
`lacde` Length dependence of acde
`lmoin` Length dependence of moin
`lnoff` Length dependence of noff
`lvoffcv` Length dependence of voffcv
`wcdsc` Width dependence of cdsc
`wcdscb` Width dependence of cdscb
`wcdscd` Width dependence of cdscd
`wcit` Width dependence of cit
`wnfactor` Width dependence of nfactor
`wxj` Width dependence of xj
`wvsat` Width dependence of vsat
`wat` Width dependence of at
`wa0` Width dependence of a0
`wags` Width dependence of ags
`wa1` Width dependence of a1
`wa2` Width dependence of a2
`wketa` Width dependence of keta
`wnsub` Width dependence of nsub
`wnch` Width dependence of nch
`wngate` Width dependence of ngate
`wgamma1` Width dependence of gamma1
`wgamma2` Width dependence of gamma2
`wvbx` Width dependence of vbx
`wvbm` Width dependence of vbm
`wxt` Width dependence of xt
`wk1` Width dependence of k1
`wkt1` Width dependence of kt1
`wkt1l` Width dependence of kt1l
`wkt2` Width dependence of kt2

`wk2` Width dependence of k2
`wk3` Width dependence of k3
`wk3b` Width dependence of k3b
`ww0` Width dependence of w0
`wnlx` Width dependence of nlx
`wdvt0` Width dependence of dvt0
`wdvt1` Width dependence of dvt1
`wdvt2` Width dependence of dvt2
`wdvt0w` Width dependence of dvt0w
`wdvt1w` Width dependence of dvt1w
`wdvt2w` Width dependence of dvt2w
`wdrout` Width dependence of drout
`wdsub` Width dependence of dsub
`wvtho` Width dependence of vto
`wvth0` Width dependence of vto
`wua` Width dependence of ua
`wua1` Width dependence of ua1
`wub` Width dependence of ub
`wub1` Width dependence of ub1
`wuc` Width dependence of uc
`wuc1` Width dependence of uc1
`wu0` Width dependence of u0
`wute` Width dependence of ute
`wvoff` Width dependence of voff
`welm` Width dependence of elm
`wdelta` Width dependence of delta
`wrdsw` Width dependence of rdsw
`wprwg` Width dependence of prwg
`wprwb` Width dependence of prwb
`wprt` Width dependence of prt
`weta0` Width dependence of eta0
`wetab` Width dependence of etab
`wpclm` Width dependence of pclm
`wpdiblc1` Width dependence of pdiblc1
`wpdiblc2` Width dependence of pdiblc2
`wpdiblcb` Width dependence of pdiblcb
`wpscbe1` Width dependence of pscbe1
`wpscbe2` Width dependence of pscbe2
`wpvag` Width dependence of pvag
`wwr` Width dependence of wr
`wdwg` Width dependence of dwg
`wdwb` Width dependence of dwb
`wb0` Width dependence of b0
`wb1` Width dependence of b1
`wcgsl` Width dependence of cgsl
`wcgdl` Width dependence of cgdl
`wckappa` Width dependence of ckappa
`wcf` Width dependence of cf
`wclc` Width dependence of clc

`wcle` Width dependence of cle
`walpha0` Width dependence of alpha0
`walpha1` Width dependence of alpha1
`wbeta0` Width dependence of beta0
`wvfbcv` Width dependence of vfbcv
`wvfb` Width dependence of vfb
`wacde` Width dependence of acde
`wmoin` Width dependence of moin
`wnoff` Width dependence of noff
`wvoffcv` Width dependence of voffcv
`pcdsc` Cross-term dependence of cdsc
`pcdscb` Cross-term dependence of cdscb
`pcdscd` Cross-term dependence of cdscd
`pcit` Cross-term dependence of cit
`pnfactor` Cross-term dependence of nfactor
`pxj` Cross-term dependence of xj
`pvsat` Cross-term dependence of vsat
`pat` Cross-term dependence of at
`pa0` Cross-term dependence of a0
`pags` Cross-term dependence of ags
`pa1` Cross-term dependence of a1
`pa2` Cross-term dependence of a2
`pketa` Cross-term dependence of keta
`pnsub` Cross-term dependence of nsub
`pnch` Cross-term dependence of nch
`pngate` Cross-term dependence of ngate
`pgamma1` Cross-term dependence of gamma1
`pgamma2` Cross-term dependence of gamma2
`pvbx` Cross-term dependence of vbx
`pvbm` Cross-term dependence of vbm
`pxt` Cross-term dependence of xt
`pk1` Cross-term dependence of k1
`pkt1` Cross-term dependence of kt1
`pkt1l` Cross-term dependence of kt1l
`pkt2` Cross-term dependence of kt2
`pk2` Cross-term dependence of k2
`pk3` Cross-term dependence of k3
`pk3b` Cross-term dependence of k3b
`pw0` Cross-term dependence of w0
`pnlx` Cross-term dependence of nlx
`pdvt0` Cross-term dependence of dvt0
`pdvt1` Cross-term dependence of dvt1
`pdvt2` Cross-term dependence of dvt2
`pdvt0w` Cross-term dependence of dvt0w
`pdvt1w` Cross-term dependence of dvt1w
`pdvt2w` Cross-term dependence of dvt2w
`pdrout` Cross-term dependence of drout
`pdsub` Cross-term dependence of dsub
`pvtho` Cross-term dependence of vto

`pvth0` Cross-term dependence of vto
`pua` Cross-term dependence of ua
`pua1` Cross-term dependence of ua1
`pub` Cross-term dependence of ub
`pub1` Cross-term dependence of ub1
`puc` Cross-term dependence of uc
`puc1` Cross-term dependence of uc1
`pu0` Cross-term dependence of u0
`pute` Cross-term dependence of ute
`pvoff` Cross-term dependence of voff
`pelm` Cross-term dependence of elm
`pdelta` Cross-term dependence of delta
`prdsw` Cross-term dependence of rdsw
`pprwg` Cross-term dependence of prwg
`pprwb` Cross-term dependence of prwb
`pprt` Cross-term dependence of prt
`peta0` Cross-term dependence of eta0
`petab` Cross-term dependence of etab
`ppclm` Cross-term dependence of pclm
`ppdiblc1` Cross-term dependence of pdiblc1
`ppdiblc2` Cross-term dependence of pdiblc2
`ppdiblcb` Cross-term dependence of pdiblcb
`ppscbe1` Cross-term dependence of pscbe1
`ppscbe2` Cross-term dependence of pscbe2
`ppvag` Cross-term dependence of pvag
`pwr` Cross-term dependence of wr
`pdwg` Cross-term dependence of dwg
`pdwb` Cross-term dependence of dwb
`pb0` Cross-term dependence of b0
`pb1` Cross-term dependence of b1
`pcgsl` Cross-term dependence of cgsl
`pcgdl` Cross-term dependence of cgdl
`pckappa` Cross-term dependence of ckappa
`pcf` Cross-term dependence of cf
`pclc` Cross-term dependence of clc
`pcle` Cross-term dependence of cle
`palpha0` Cross-term dependence of alpha0
`palpha1` Cross-term dependence of alpha1
`pbeta0` Cross-term dependence of beta0
`pvfbcv` Cross-term dependence of vfbcv
`pvfb` Cross-term dependence of vfb
`pacde` Cross-term dependence of acde
`pmoin` Cross-term dependence of moin
`pnoff` Cross-term dependence of noff
`pvoffcv` Cross-term dependence of voffcv
`noia` Flicker noise parameter
`noib` Flicker noise parameter
`noic` Flicker noise parameter
`em` Flicker noise parameter

`ef` Flicker noise frequency exponent
`af` Flicker noise exponent
`kf` Flicker noise coefficient
`cluster` List (-): list of bsim3v3 models composing the cluster. A cluster does not describe a bsim3v3 model itself, but groups of already defined bsim3v3 models. Each model in the list is characterised by the <wmin>, <wmax>, <lmin> and <lmax> parameters. When an instance refers to a cluster of models, the list is searched and the current width and length of the instance are compared to those of the models in the list. When the width and lenght of the instance fall inbetween the min/max values specified in one model of the list, it is bound to the instance. To summarise, clusters are a way to choose models by using width and length of the given instance as a selection criterion.
`bounderror` Integer (2): the severity of the error given when <l> and/or <w> are out of bounds (<lmin>/<lmax> or <wmin>/<wmax>). Possible choises are: 0: none (error is not given) 1: warning (simulation continues) 2: error (simulation is aborted). Note that if this parameter is specified it overwrites the value of the corresponding parameter given among the <options> of the simulator.
`acm` Area calculation method selector
`hdif` ACM Parameter: Distance Gate - contact
`ldif` ACM Parameter: Length of LDD Gate-Source/Drain
`ld` ACM Parameter: Length of LDD under Gate
`paramchk` Model parameter checking selector
`rd` ACM Parameter: Resistance of LDD drain side
`rs` ACM Parameter: Resistance of LDD source side
`rdc` ACM Parameter: Resistance contact drain side
`rsc` ACM Parameter: Resistance contact source side
`version` String ("3.2.2"): model version. Supported ones are "3.2.4", "3.2.3", "3.2.2", "3.2".
`xl` Length correction parameter
`xw` Width correction parameter

`saref` Reference distance between od edge to poly of one side.
`sbref` Reference distance between od edge to poly of one other side.
`kvth0` Reference distance between od edge to poly of one other side.
`lkvth0` Threshold shift parameter for stress effect.
`stk2` k2 shift factor related to vth0 change.
`lku0` Length dependence of ku0.
`ku0` Mobility degradation/enhancement coefficient for stress effect.
`tku0` Temperature coefficient of ku0.
`kvsat` saturation velocity degradation/enhancement parameter for stress effect.
`steta0` Eta0 shift factor related to vth0 change.
`tlev` DC temperature selector
`tlevc` DC temperature selector
`type` String (-): the mosfet type. Available keywords are <n> and <p>

4.4.8 Berkeley Short Channel IGFET Model Version-4.50

ELEMENT DEFINITION:
Name drain gate source bulk **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`l` Length

`w` Width
`m` Integer (1.0): number of devices connected in parallel.
`nf` Number of fingers
`sa` distance between OD edge to poly of one side
`sb` distance between OD edge to poly of the other side
`sd` distance between neighbour fingers
`sca` Integral of the first distribution function for scattered well dopant
`scb` Integral of the second distribution function for scattered well dopant
`scc` Integral of the third distribution function for scattered well dopant
`sc` Distance to a single well edge
`min` Minimize either D or S
`ad` Drain area
`as` Source area
`pd` Drain perimeter
`ps` Source perimeter
`nrd` Number of squares in drain
`nrs` Number of squares in source
`rbdb` Body resistance
`rbsb` Body resistance
`rbpb` Body resistance
`rbps` Body resistance
`rbpd` Body resistance
`delvto` Zero bias threshold voltage variation
`xgw` Distance from gate contact center to device edge
`ngcon` Number of gate contacts
`trnqsmod` Transient NQS model selector
`acnqsmod` AC NQS model selector
`rbodymod` Distributed body R model selector
`rgatemod` Gate resistance model selector
`geomod` Geometry dependent parasitics model selector
`rgeomod` S/D resistance and contact model selector
`temp` Device junction temperature
`noisetype` Integer (7): types of generated noise; possible choices are:
0x0 : noiseless BSIM4; 0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

MODEL DEFINITION:

model *ModelName* **bsim4** <*parameter*=*value*> ...

MODEL PARAMETERS:

`capmod` Capacitance model selector
`diomod` Diode IV model selector
`rdsmod` Bias-dependent S/D resistance model selector
`trnqsmod` Transient NQS model selector
`acnqsmod` AC NQS model selector
`mobmod` Mobility model selector
`rbodymod` Distributed body R model selector
`rgatemod` Gate R model selector
`permod` Pd and Ps model selector
`geomod` Geometry dependent parasitics model selector

`fnoimod` Flicker noise model selector
`tnoimod` Thermal noise model selector
`igcmod` Gate-to-channel Ig model selector
`igbmod` Gate-to-body Ig model selector
`tempmod` Temperature model selector
`paramchk` Model parameter checking selector
`binunit` Bin unit selector
`version` parameter for model version
`toxe` Electrical gate oxide thickness in meters
`toxp` Physical gate oxide thickness in meters
`toxm` Gate oxide thickness at which parameters are extracted
`toxref` Target tox value
`dtox` Defined as (toxe - toxp
`epsrox` Dielectric constant of the gate oxide relative to vacuum
`cdsc` Drain/Source and channel coupling capacitance
`cdscb` Body-bias dependence of cdsc
`cdscd` Drain-bias dependence of cdsc
`cit` Interface state capacitance
`nfactor` Subthreshold swing Coefficient
`xj` Junction depth in meters
`vsat` Saturation velocity at tnom
`at` Temperature coefficient of vsat
`a0` Non-uniform depletion width effect coefficient.
`ags` Gate bias coefficient of Abulk.
`a1` Non-saturation effect coefficient
`a2` Non-saturation effect coefficient
`keta` Body-bias coefficient of non-uniform depletion width effect.
`nsub` Substrate doping concentration
`ndep` Channel doping concentration at the depletion edge
`nsd` S/D doping concentration
`phin` Adjusting parameter for surface potential due to non-uniform vertical doping
`ngate` Poly-gate doping concentration
`gamma1` Vth body coefficient
`gamma2` Vth body coefficient
`vbx` Vth transition body Voltage
`vbm` Maximum body voltage
`xt` Doping depth
`k1` Bulk effect coefficient 1
`kt1` Temperature coefficient of Vth
`kt1l` Temperature coefficient of Vth
`kt2` Body-coefficient of kt1
`k2` Bulk effect coefficient 2
`k3` Narrow width effect coefficient
`k3b` Body effect coefficient of k3
`w0` Narrow width effect parameter
`dvtp0` First parameter for Vth shift due to pocket
`dvtp1` Second parameter for Vth shift due to pocket
`lpe0` Equivalent length of pocket region at zero bias
`lpeb` Equivalent length of pocket region accounting for body bias

`dvt0` Short channel effect coeff. 0
`dvt1` Short channel effect coeff. 1
`dvt2` Short channel effect coeff. 2
`dvt0w` Narrow Width coeff. 0
`dvt1w` Narrow Width effect coeff. 1
`dvt2w` Narrow Width effect coeff. 2
`drout` DIBL coefficient of output resistance
`dsub` DIBL coefficient in the subthreshold region
`vth0` Threshold voltage
`vtho` Threshold voltage
`ua` Linear gate dependence of mobility
`ua1` Temperature coefficient of ua
`ub` Quadratic gate dependence of mobility
`ub1` Temperature coefficient of ub
`uc` Body-bias dependence of mobility
`uc1` Temperature coefficient of uc
`ud` Coulomb scattering factor of mobility
`ud1` Temperature coefficient of ud
`up` Channel length linear factor of mobility
`lp` Channel length exponential factor of mobility
`u0` Low-field mobility at Tnom
`eu` Mobility exponent
`ute` Temperature coefficient of mobility
`voff` Threshold voltage offset
`minv` Fitting parameter for moderate invversion in Vgsteff
`voffl` Length dependence parameter for Vth offset
`tnom` Parameter measurement temperature
`cgso` Gate-source overlap capacitance per width
`cgdo` Gate-drain overlap capacitance per width
`cgbo` Gate-bulk overlap capacitance per length
`xpart` Channel charge partitioning
`delta` Effective Vds parameter
`rsh` Source-drain sheet resistance
`rdsw` Source-drain resistance per width
`rdswmin` Source-drain resistance per width at high Vg
`rsw` Source resistance per width
`rdw` Drain resistance per width
`rdwmin` Drain resistance per width at high Vg
`rswmin` Source resistance per width at high Vg
`prwg` Gate-bias effect on parasitic resistance
`prwb` Body-effect on parasitic resistance
`prt` Temperature coefficient of parasitic resistance
`eta0` Subthreshold region DIBL coefficient
`etab` Subthreshold region DIBL coefficient
`pclm` Channel length modulation Coefficient
`pdiblc1` Drain-induced barrier lowering coefficient
`pdiblc2` Drain-induced barrier lowering coefficient
`pdiblcb` Body-effect on drain-induced barrier lowering
`fprout` Rout degradation coefficient for pocket devices

`pdits` Coefficient for drain-induced Vth shifts
`pditsl` Length dependence of drain-induced Vth shifts
`pditsd` Vds dependence of drain-induced Vth shifts
`pscbe1` Substrate current body-effect coefficient
`pscbe2` Substrate current body-effect coefficient
`pvag` Gate dependence of output resistance parameter
`jss` Bottom source junction reverse saturation current density
`jsws` Isolation edge sidewall source junction reverse saturation current density
`jswgs` Gate edge source junction reverse saturation current density
`pbs` Source junction built-in potential
`njs` Source junction emission coefficient
`xtis` Source junction current temperature exponent
`mjs` Source bottom junction capacitance grading coefficient
`pbsws` Source sidewall junction capacitance built in potential
`mjsws` Source sidewall junction capacitance grading coefficient
`pbswgs` Source (gate side sidewall junction capacitance built in potential
`mjswgs` Source (gate side sidewall junction capacitance grading coefficient
`cjs` Source bottom junction capacitance per unit area
`cjsws` Source sidewall junction capacitance per unit periphery
`cjswgs` Source (gate side sidewall junction capacitance per unit width
`jsd` Bottom drain junction reverse saturation current density
`jswd` Isolation edge sidewall drain junction reverse saturation current density
`jswgd` Gate edge drain junction reverse saturation current density
`pbd` Drain junction built-in potential
`njd` Drain junction emission coefficient
`xtid` Drainjunction current temperature exponent
`mjd` Drain bottom junction capacitance grading coefficient
`pbswd` Drain sidewall junction capacitance built in potential
`mjswd` Drain sidewall junction capacitance grading coefficient
`pbswgd` Drain (gate side sidewall junction capacitance built in potential
`mjswgd` Drain (gate side sidewall junction capacitance grading coefficient
`cjd` Drain bottom junction capacitance per unit area
`cjswd` Drain sidewall junction capacitance per unit periphery
`cjswgd` Drain (gate side sidewall junction capacitance per unit width
`vfbcv` Flat Band Voltage parameter for capmod=0 only
`vfb` Flat Band Voltage
`tpb` Temperature coefficient of pb
`tcj` Temperature coefficient of cj
`tpbsw` Temperature coefficient of pbsw
`tcjsw` Temperature coefficient of cjsw
`tpbswg` Temperature coefficient of pbswg
`tcjswg` Temperature coefficient of cjswg
`acde` Exponential coefficient for finite charge thickness
`moin` Coefficient for gate-bias dependent surface potential
`noff` C-V turn-on/off parameter
`voffcv` C-V lateral-shift parameter
`dmcg` Distance of Mid-Contact to Gate edge
`dmci` Distance of Mid-Contact to Isolation
`dmdg` Distance of Mid-Diffusion to Gate edge

`dmcgt` Distance of Mid-Contact to Gate edge in Test structures
`xgw` Distance from gate contact center to device edge
`xgl` Variation in Ldrawn
`rshg` Gate sheet resistance
`ngcon` Number of gate contacts
`xrcrg1` First fitting parameter the bias-dependent Rg
`xrcrg2` Second fitting parameter the bias-dependent Rg
`lambda` Velocity overshoot parameter
`vtl` thermal velocity
`lc` back scattering parameter
`xn` back scattering parameter
`vfbsdoff` S/D flatband voltage offset
`tvfbsdoff` Temperature parameter for vfbsdoff
`tvoff` Temperature parameter for voff
`lintnoi` lint offset for noise calculation
`lint` Length reduction parameter
`ll` Length reduction parameter
`llc` Length reduction parameter for CV
`lln` Length reduction parameter
`lw` Length reduction parameter
`lwc` Length reduction parameter for CV
`lwn` Length reduction parameter
`lwl` Length reduction parameter
`lwlc` Length reduction parameter for CV
`lmin` Minimum length for the model
`lmax` Maximum length for the model
`wr` Width dependence of rds
`wint` Width reduction parameter
`dwg` Width reduction parameter
`dwb` Width reduction parameter
`wl` Width reduction parameter
`wlc` Width reduction parameter for CV
`wln` Width reduction parameter
`ww` Width reduction parameter
`wwc` Width reduction parameter for CV
`wwn` Width reduction parameter
`wwl` Width reduction parameter
`wwlc` Width reduction parameter for CV
`wmin` Minimum width for the model
`wmax` Maximum width for the model
`b0` Abulk narrow width parameter
`b1` Abulk narrow width parameter
`cgsl` New C-V model parameter
`cgdl` New C-V model parameter
`ckappas` S/G overlap C-V parameter
`ckappad` D/G overlap C-V parameter
`cf` Fringe capacitance parameter
`clc` Vdsat parameter for C-V model
`cle` Vdsat parameter for C-V model

`dwc` Delta W for C-V model
`dlc` Delta L for C-V model
`xw` W offset for channel width due to mask/etch effect
`xl` L offset for channel length due to mask/etch effect
`dlcig` Delta L for Ig model
`dwj` Delta W for S/D junctions
`alpha0` substrate current model parameter
`alpha1` substrate current model parameter
`beta0` substrate current model parameter
`agidl` Pre-exponential constant for GIDL
`bgidl` Exponential constant for GIDL
`cgidl` Parameter for body-bias dependence of GIDL
`egidl` Fitting parameter for Bandbending
`aigc` Parameter for Igc
`bigc` Parameter for Igc
`cigc` Parameter for Igc
`aigsd` Parameter for Igs
`bigsd` Parameter for Igs
`cigsd` Parameter for Igs
`aigbacc` Parameter for Igb
`bigbacc` Parameter for Igb
`cigbacc` Parameter for Igb
`aigbinv` Parameter for Igb
`bigbinv` Parameter for Igb
`cigbinv` Parameter for Igb
`nigc` Parameter for Igc slope
`nigbinv` Parameter for Igbinv slope
`nigbacc` Parameter for Igbacc slope
`ntox` Exponent for Tox ratio
`eigbinv` Parameter for the Si bandgap for Igbinv
`pigcd` Parameter for Igc partition
`poxedge` Factor for the gate edge Tox
`ijthdfwd` Forward drain diode forward limiting current
`ijthsfwd` Forward source diode forward limiting current
`ijthdrev` Reverse drain diode forward limiting current
`ijthsrev` Reverse source diode forward limiting current
`xjbvd` Fitting parameter for drain diode breakdown current
`xjbvs` Fitting parameter for source diode breakdown current
`bvd` Drain diode breakdown voltage
`bvs` Source diode breakdown voltage
`jtss` Source bottom trap-assisted saturation current density
`jtsd` Drain bottom trap-assisted saturation current density
`jtssws` Source STI sidewall trap-assisted saturation current density
`jtsswd` Drain STI sidewall trap-assisted saturation current density
`jtsswgs` Source gate-edge sidewall trap-assisted saturation current density
`jtsswgd` Drain gate-edge sidewall trap-assisted saturation current density
`njts` Non-ideality factor for bottom junction
`njtssw` Non-ideality factor for STI sidewall junction
`njtsswg` Non-ideality factor for gate-edge sidewall junction

`xtss` Power dependence of JTSS on temperature
`xtsd` Power dependence of JTSD on temperature
`xtssws` Power dependence of JTSSWS on temperature
`xtsswd` Power dependence of JTSSWD on temperature
`xtsswgs` Power dependence of JTSSWGS on temperature
`xtsswgd` Power dependence of JTSSWGD on temperature
`tnjts` Temperature coefficient for NJTS
`tnjtssw` Temperature coefficient for NJTSSW
`tnjtsswg` Temperature coefficient for NJTSSWG
`vtss` Source bottom trap-assisted voltage dependent parameter
`vtsd` Drain bottom trap-assisted voltage dependent parameter
`vtssws` Source STI sidewall trap-assisted voltage dependent parameter
`vtsswd` Drain STI sidewall trap-assisted voltage dependent parameter
`vtsswgs` Source gate-edge sidewall trap-assisted voltage dependent parameter
`vtsswgd` Drain gate-edge sidewall trap-assisted voltage dependent parameter
`gbmin` Minimum body conductance
`rbdb` Resistance between bNode and dbNode
`rbpb` Resistance between bNodePrime and bNode
`rbsb` Resistance between bNode and sbNode
`rbps` Resistance between bNodePrime and sbNode
`rbpd` Resistance between bNodePrime and bNode
`rbps0` Body resistance RBPS scaling
`rbpsl` Body resistance RBPS L scaling
`rbpsw` Body resistance RBPS W scaling
`rbpsnf` Body resistance RBPS NF scaling
`rbpd0` Body resistance RBPD scaling
`rbpdl` Body resistance RBPD L scaling
`rbpdw` Body resistance RBPD W scaling
`rbpdnf` Body resistance RBPD NF scaling
`rbpbx0` Body resistance RBPBX scaling
`rbpbxl` Body resistance RBPBX L scaling
`rbpbxw` Body resistance RBPBX W scaling
`rbpbxnf` Body resistance RBPBX NF scaling
`rbpby0` Body resistance RBPBY scaling
`rbpbyl` Body resistance RBPBY L scaling
`rbpbyw` Body resistance RBPBY W scaling
`rbpbynf` Body resistance RBPBY NF scaling
`rbsbx0` Body resistance RBSBX scaling
`rbsby0` Body resistance RBSBY scaling
`rbdbx0` Body resistance RBDBX scaling
`rbdby0` Body resistance RBDBY scaling
`rbsdbxl` Body resistance RBSDBX L scaling
`rbsdbxw` Body resistance RBSDBX W scaling
`rbsdbxnf` Body resistance RBSDBX NF scaling
`rbsdbyl` Body resistance RBSDBY L scaling
`rbsdbyw` Body resistance RBSDBY W scaling
`rbsdbynf` Body resistance RBSDBY NF scaling
`lcdsc` Length dependence of cdsc
`lcdscb` Length dependence of cdscb

`lcdscd` Length dependence of cdscd
`lcit` Length dependence of cit
`lnfactor` Length dependence of nfactor
`lxj` Length dependence of xj
`lvsat` Length dependence of vsat
`lat` Length dependence of at
`la0` Length dependence of a0
`lags` Length dependence of ags
`la1` Length dependence of a1
`la2` Length dependence of a2
`lketa` Length dependence of keta
`lnsub` Length dependence of nsub
`lndep` Length dependence of ndep
`lnsd` Length dependence of nsd
`lphin` Length dependence of phin
`lngate` Length dependence of ngate
`lgamma1` Length dependence of gamma1
`lgamma2` Length dependence of gamma2
`lvbx` Length dependence of vbx
`lvbm` Length dependence of vbm
`lxt` Length dependence of xt
`lk1` Length dependence of k1
`lkt1` Length dependence of kt1
`lkt1l` Length dependence of kt1l
`lkt2` Length dependence of kt2
`lk2` Length dependence of k2
`lk3` Length dependence of k3
`lk3b` Length dependence of k3b
`lw0` Length dependence of w0
`ldvtp0` Length dependence of dvtp0
`ldvtp1` Length dependence of dvtp1
`llpe0` Length dependence of lpe0
`llpeb` Length dependence of lpeb
`ldvt0` Length dependence of dvt0
`ldvt1` Length dependence of dvt1
`ldvt2` Length dependence of dvt2
`ldvt0w` Length dependence of dvt0w
`ldvt1w` Length dependence of dvt1w
`ldvt2w` Length dependence of dvt2w
`ldrout` Length dependence of drout
`ldsub` Length dependence of dsub
`lvth0` Length dependence of vto
`lvtho` Length dependence of vto
`lua` Length dependence of ua
`lua1` Length dependence of ua1
`lub` Length dependence of ub
`lub1` Length dependence of ub1
`luc` Length dependence of uc
`luc1` Length dependence of uc1

`lud` Length dependence of ud
`lud1` Length dependence of ud1
`lup` Length dependence of up
`llp` Length dependence of lp
`lu0` Length dependence of u0
`lute` Length dependence of ute
`lvoff` Length dependence of voff
`lminv` Length dependence of minv
`ldelta` Length dependence of delta
`lrdsw` Length dependence of rdsw
`lrsw` Length dependence of rsw
`lrdw` Length dependence of rdw
`lprwg` Length dependence of prwg
`lprwb` Length dependence of prwb
`lprt` Length dependence of prt
`leta0` Length dependence of eta0
`letab` Length dependence of etab
`lpclm` Length dependence of pclm
`lpdiblc1` Length dependence of pdiblc1
`lpdiblc2` Length dependence of pdiblc2
`lpdiblcb` Length dependence of pdiblcb
`lfprout` Length dependence of pdiblcb
`lpdits` Length dependence of pdits
`lpditsd` Length dependence of pditsd
`lpscbe1` Length dependence of pscbe1
`lpscbe2` Length dependence of pscbe2
`lpvag` Length dependence of pvag
`lwr` Length dependence of wr
`ldwg` Length dependence of dwg
`ldwb` Length dependence of dwb
`lb0` Length dependence of b0
`lb1` Length dependence of b1
`lcgsl` Length dependence of cgsl
`lcgdl` Length dependence of cgdl
`lckappas` Length dependence of ckappas
`lckappad` Length dependence of ckappad
`lcf` Length dependence of cf
`lclc` Length dependence of clc
`lcle` Length dependence of cle
`lalpha0` Length dependence of alpha0
`lalpha1` Length dependence of alpha1
`lbeta0` Length dependence of beta0
`lagidl` Length dependence of agidl
`lbgidl` Length dependence of bgidl
`lcgidl` Length dependence of cgidl
`legidl` Length dependence of egidl
`laigc` Length dependence of aigc
`lbigc` Length dependence of bigc
`lcigc` Length dependence of cigc

`laigsd` Length dependence of aigsd
`lbigsd` Length dependence of bigsd
`lcigsd` Length dependence of cigsd
`laigbacc` Length dependence of aigbacc
`lbigbacc` Length dependence of bigbacc
`lcigbacc` Length dependence of cigbacc
`laigbinv` Length dependence of aigbinv
`lbigbinv` Length dependence of bigbinv
`lcigbinv` Length dependence of cigbinv
`lnigc` Length dependence of nigc
`lnigbinv` Length dependence of nigbinv
`lnigbacc` Length dependence of nigbacc
`lntox` Length dependence of ntox
`leigbinv` Length dependence for eigbinv
`lpigcd` Length dependence for pigcd
`lpoxedge` Length dependence for poxedge
`lvfbcv` Length dependence of vfbcv
`lvfb` Length dependence of vfb
`lacde` Length dependence of acde
`lmoin` Length dependence of moin
`lnoff` Length dependence of noff
`lvoffcv` Length dependence of voffcv
`lxrcrg1` Length dependence of xrcrg1
`lxrcrg2` Length dependence of xrcrg2
`llambda` Length dependence of lambda
`lvtl` Length dependence of vtl
`lxn` Length dependence of xn
`leu` Length dependence of eu
`lvfbsdoff` Length dependence of vfbsdoff
`ltvfbsdoff` Length dependence of tvfbsdoff
`ltvoff` Length dependence of tvoff
`wcdsc` Width dependence of cdsc
`wcdscb` Width dependence of cdscb
`wcdscd` Width dependence of cdscd
`wcit` Width dependence of cit
`wnfactor` Width dependence of nfactor
`wxj` Width dependence of xj
`wvsat` Width dependence of vsat
`wat` Width dependence of at
`wa0` Width dependence of a0
`wags` Width dependence of ags
`wa1` Width dependence of a1
`wa2` Width dependence of a2
`wketa` Width dependence of keta
`wnsub` Width dependence of nsub
`wndep` Width dependence of ndep
`wnsd` Width dependence of nsd
`wphin` Width dependence of phin
`wngate` Width dependence of ngate

`wgamma1` Width dependence of gamma1
`wgamma2` Width dependence of gamma2
`wvbx` Width dependence of vbx
`wvbm` Width dependence of vbm
`wxt` Width dependence of xt
`wk1` Width dependence of k1
`wkt1` Width dependence of kt1
`wkt1l` Width dependence of kt1l
`wkt2` Width dependence of kt2
`wk2` Width dependence of k2
`wk3` Width dependence of k3
`wk3b` Width dependence of k3b
`ww0` Width dependence of w0
`wdvtp0` Width dependence of dvtp0
`wdvtp1` Width dependence of dvtp1
`wlpe0` Width dependence of lpe0
`wlpeb` Width dependence of lpeb
`wdvt0` Width dependence of dvt0
`wdvt1` Width dependence of dvt1
`wdvt2` Width dependence of dvt2
`wdvt0w` Width dependence of dvt0w
`wdvt1w` Width dependence of dvt1w
`wdvt2w` Width dependence of dvt2w
`wdrout` Width dependence of drout
`wdsub` Width dependence of dsub
`wvth0` Width dependence of vto
`wvth0` Width dependence of vto
`wua` Width dependence of ua
`wua1` Width dependence of ua1
`wub` Width dependence of ub
`wub1` Width dependence of ub1
`wuc` Width dependence of uc
`wuc1` Width dependence of uc1
`wud` Width dependence of ud
`wud1` Width dependence of ud1
`wup` Width dependence of up
`wlp` Width dependence of lp
`wu0` Width dependence of u0
`wute` Width dependence of ute
`wvoff` Width dependence of voff
`wminv` Width dependence of minv
`wdelta` Width dependence of delta
`wrdsw` Width dependence of rdsw
`wrsw` Width dependence of rsw
`wrdw` Width dependence of rdw
`wprwg` Width dependence of prwg
`wprwb` Width dependence of prwb
`wprt` Width dependence of prt
`weta0` Width dependence of eta0

`wetab` Width dependence of etab
`wpclm` Width dependence of pclm
`wpdiblc1` Width dependence of pdiblc1
`wpdiblc2` Width dependence of pdiblc2
`wpdiblcb` Width dependence of pdiblcb
`wfprout` Width dependence of pdiblcb
`wpdits` Width dependence of pdits
`wpditsd` Width dependence of pditsd
`wpscbe1` Width dependence of pscbe1
`wpscbe2` Width dependence of pscbe2
`wpvag` Width dependence of pvag
`wwr` Width dependence of wr
`wdwg` Width dependence of dwg
`wdwb` Width dependence of dwb
`wb0` Width dependence of b0
`wb1` Width dependence of b1
`wcgsl` Width dependence of cgsl
`wcgdl` Width dependence of cgdl
`wckappas` Width dependence of ckappas
`wckappad` Width dependence of ckappad
`wcf` Width dependence of cf
`wclc` Width dependence of clc
`wcle` Width dependence of cle
`walpha0` Width dependence of alpha0
`walpha1` Width dependence of alpha1
`wbeta0` Width dependence of beta0
`wagidl` Width dependence of agidl
`wbgidl` Width dependence of bgidl
`wcgidl` Width dependence of cgidl
`wegidl` Width dependence of egidl
`waigc` Width dependence of aigc
`wbigc` Width dependence of bigc
`wcigc` Width dependence of cigc
`waigsd` Width dependence of aigsd
`wbigsd` Width dependence of bigsd
`wcigsd` Width dependence of cigsd
`waigbacc` Width dependence of aigbacc
`wbigbacc` Width dependence of bigbacc
`wcigbacc` Width dependence of cigbacc
`waigbinv` Width dependence of aigbinv
`wbigbinv` Width dependence of bigbinv
`wcigbinv` Width dependence of cigbinv
`wnigc` Width dependence of nigc
`wnigbinv` Width dependence of nigbinv
`wnigbacc` Width dependence of nigbacc
`wntox` Width dependence of ntox
`weigbinv` Width dependence for eigbinv
`wpigcd` Width dependence for pigcd
`wpoxedge` Width dependence for poxedge

`wvfbcv` Width dependence of vfbcv
`wvfb` Width dependence of vfb
`wacde` Width dependence of acde
`wmoin` Width dependence of moin
`wnoff` Width dependence of noff
`wvoffcv` Width dependence of voffcv
`wxrcrg1` Width dependence of xrcrg1
`wxrcrg2` Width dependence of xrcrg2
`wlambda` Width dependence of lambda
`wvtl` Width dependence of vtl
`wxn` Width dependence of xn
`weu` Width dependence of eu
`wvfbsdoff` Width dependence of vfbsdoff
`wtvfbsdoff` Width dependence of tvfbsdoff
`wtvoff` Width dependence of tvoff
`pcdsc` Cross-term dependence of cdsc
`pcdscb` Cross-term dependence of cdscb
`pcdscd` Cross-term dependence of cdscd
`pcit` Cross-term dependence of cit
`pnfactor` Cross-term dependence of nfactor
`pxj` Cross-term dependence of xj
`pvsat` Cross-term dependence of vsat
`pat` Cross-term dependence of at
`pa0` Cross-term dependence of a0
`pags` Cross-term dependence of ags
`pa1` Cross-term dependence of a1
`pa2` Cross-term dependence of a2
`pketa` Cross-term dependence of keta
`pnsub` Cross-term dependence of nsub
`pndep` Cross-term dependence of ndep
`pnsd` Cross-term dependence of nsd
`pphin` Cross-term dependence of phin
`pngate` Cross-term dependence of ngate
`pgamma1` Cross-term dependence of gamma1
`pgamma2` Cross-term dependence of gamma2
`pvbx` Cross-term dependence of vbx
`pvbm` Cross-term dependence of vbm
`pxt` Cross-term dependence of xt
`pk1` Cross-term dependence of k1
`pkt1` Cross-term dependence of kt1
`pkt1l` Cross-term dependence of kt1l
`pkt2` Cross-term dependence of kt2
`pk2` Cross-term dependence of k2
`pk3` Cross-term dependence of k3
`pk3b` Cross-term dependence of k3b
`pw0` Cross-term dependence of w0
`pdvtp0` Cross-term dependence of dvtp0
`pdvtp1` Cross-term dependence of dvtp1
`plpe0` Cross-term dependence of lpe0

`plpeb` Cross-term dependence of lpeb
`pdvt0` Cross-term dependence of dvt0
`pdvt1` Cross-term dependence of dvt1
`pdvt2` Cross-term dependence of dvt2
`pdvt0w` Cross-term dependence of dvt0w
`pdvt1w` Cross-term dependence of dvt1w
`pdvt2w` Cross-term dependence of dvt2w
`pdrout` Cross-term dependence of drout
`pdsub` Cross-term dependence of dsub
`pvth0` Cross-term dependence of vto
`pvtho` Cross-term dependence of vto
`pua` Cross-term dependence of ua
`pua1` Cross-term dependence of ua1
`pub` Cross-term dependence of ub
`pub1` Cross-term dependence of ub1
`puc` Cross-term dependence of uc
`puc1` Cross-term dependence of uc1
`pud` Cross-term dependence of ud
`pud1` Cross-term dependence of ud1
`pup` Cross-term dependence of up
`plp` Cross-term dependence of lp
`pu0` Cross-term dependence of u0
`pute` Cross-term dependence of ute
`pvoff` Cross-term dependence of voff
`pminv` Cross-term dependence of minv
`pdelta` Cross-term dependence of delta
`prdsw` Cross-term dependence of rdsw
`prsw` Cross-term dependence of rsw
`prdw` Cross-term dependence of rdw
`pprwg` Cross-term dependence of prwg
`pprwb` Cross-term dependence of prwb
`pprt` Cross-term dependence of prt
`peta0` Cross-term dependence of eta0
`petab` Cross-term dependence of etab
`ppclm` Cross-term dependence of pclm
`ppdiblc1` Cross-term dependence of pdiblc1
`ppdiblc2` Cross-term dependence of pdiblc2
`ppdiblcb` Cross-term dependence of pdiblcb
`pfprout` Cross-term dependence of pdiblcb
`ppdits` Cross-term dependence of pdits
`ppditsd` Cross-term dependence of pditsd
`ppscbe1` Cross-term dependence of pscbe1
`ppscbe2` Cross-term dependence of pscbe2
`ppvag` Cross-term dependence of pvag
`pwr` Cross-term dependence of wr
`pdwg` Cross-term dependence of dwg
`pdwb` Cross-term dependence of dwb
`pb0` Cross-term dependence of b0
`pb1` Cross-term dependence of b1

`pcgsl` Cross-term dependence of cgsl
`pcgdl` Cross-term dependence of cgdl
`pckappas` Cross-term dependence of ckappas
`pckappad` Cross-term dependence of ckappad
`pcf` Cross-term dependence of cf
`pclc` Cross-term dependence of clc
`pcle` Cross-term dependence of cle
`palpha0` Cross-term dependence of alpha0
`palpha1` Cross-term dependence of alpha1
`pbeta0` Cross-term dependence of beta0
`pagidl` Cross-term dependence of agidl
`pbgidl` Cross-term dependence of bgidl
`pcgidl` Cross-term dependence of cgidl
`pegidl` Cross-term dependence of egidl
`paigc` Cross-term dependence of aigc
`pbigc` Cross-term dependence of bigc
`pcigc` Cross-term dependence of cigc
`paigsd` Cross-term dependence of aigsd
`pbigsd` Cross-term dependence of bigsd
`pcigsd` Cross-term dependence of cigsd
`paigbacc` Cross-term dependence of aigbacc
`pbigbacc` Cross-term dependence of bigbacc
`pcigbacc` Cross-term dependence of cigbacc
`paigbinv` Cross-term dependence of aigbinv
`pbigbinv` Cross-term dependence of bigbinv
`pcigbinv` Cross-term dependence of cigbinv
`pnigc` Cross-term dependence of nigc
`pnigbinv` Cross-term dependence of nigbinv
`pnigbacc` Cross-term dependence of nigbacc
`pntox` Cross-term dependence of ntox
`peigbinv` Cross-term dependence for eigbinv
`ppigcd` Cross-term dependence for pigcd
`ppoxedge` Cross-term dependence for poxedge
`pvfbcv` Cross-term dependence of vfbcv
`pvfb` Cross-term dependence of vfb
`pacde` Cross-term dependence of acde
`pmoin` Cross-term dependence of moin
`pnoff` Cross-term dependence of noff
`pvoffcv` Cross-term dependence of voffcv
`pxrcrg1` Cross-term dependence of xrcrg1
`pxrcrg2` Cross-term dependence of xrcrg2
`plambda` Cross-term dependence of lambda
`pvtl` Cross-term dependence of vtl
`pxn` Cross-term dependence of xn
`peu` Cross-term dependence of eu
`pvfbsdoff` Cross-term dependence of vfbsdoff
`ptvfbsdoff` Cross-term dependence of tvfbsdoff
`ptvoff` Cross-term dependence of tvoff
`saref` Reference distance between OD edge to poly of one side

`sbref` Reference distance between OD edge to poly of the other side
`wlod` Width parameter for stress effect
`ku0` Mobility degradation/enhancement coefficient for LOD
`kvsat` Saturation velocity degradation/enhancement parameter for LOD
`kvth0` Threshold degradation/enhancement parameter for LOD
`tku0` Temperature coefficient of KU0
`llodku0` Length parameter for u0 LOD effect
`wlodku0` Width parameter for u0 LOD effect
`llodvth` Length parameter for vth LOD effect
`wlodvth` Width parameter for vth LOD effect
`lku0` Length dependence of ku0
`wku0` Width dependence of ku0
`pku0` Cross-term dependence of ku0
`lkvth0` Length dependence of kvth0
`wkvth0` Width dependence of kvth0
`pkvth0` Cross-term dependence of kvth0
`stk2` K2 shift factor related to stress effect on vth
`lodk2` K2 shift modification factor for stress effect
`steta0` eta0 shift factor related to stress effect on vth
`lodeta0` eta0 shift modification factor for stress effect
`web` Coefficient for SCB
`wec` Coefficient for SCC
`kvth0we` Threshold shift factor for well proximity effect
`k2we` K2 shift factor for well proximity effect
`ku0we` Mobility degradation factor for well proximity effect
`scref` Reference distance to calculate SCA
`wpemod` Flag for WPE model (WPEMOD=1 to activate this model
`lkvth0we` Length dependence of kvth0we
`lk2we` Length dependence of k2we
`lku0we` Length dependence of ku0we
`wkvth0we` Width dependence of kvth0we
`wk2we` Width dependence of k2we
`wku0we` Width dependence of ku0we
`pkvth0we` Cross-term dependence of kvth0we
`pk2we` Cross-term dependence of k2we
`pku0we` Cross-term dependence of ku0we
`noia` Flicker noise parameter
`noib` Flicker noise parameter
`noic` Flicker noise parameter
`tnoia` Thermal noise parameter
`tnoib` Thermal noise parameter
`rnoia` Thermal noise coefficient
`rnoib` Thermal noise coefficient
`ntnoi` Thermal noise parameter
`em` Flicker noise parameter
`ef` Flicker noise frequency exponent
`af` Flicker noise exponent
`kf` Flicker noise coefficient
`nmos` Flag to indicate NMOS

`pmos` Flag to indicate PMOS
`cluster` List (-): list of bsim4 models composing the cluster. A cluster does not describe a bsim4 model itself, but groups of already defined bsim4 models. Each model in the list is characterised by the <wmin>, <wmax>, <lmin> and <lmax> parameters. When an instance refers to a cluster of models, the list is searched and the current width and length of the instance are compared to those of the models in the list. When the width and lenght of the instance fall inbetween the min/max values specified in one model of the list, it is bound to the instance. To summarise, clusters are a way to choose models by using width and length of the given instance as a selection criterion.
`bounderror` Integer (2): the severity of the error given when <l> and/or <w> are out of bounds (<lmin>/<lmax> or <wmin>/<wmax>). Possible choises are: 0: none (error is not given) 1: warning (simulation continues) 2: error (simulation is aborted). Note that if this parameter is specified it overwrites the value of the corresponding parameter given among the <options> of the simulator.
`type` String (-): the mosfet type. Available keywords are <n> and <p>
`vbox` Real (1.0e9*tox): oxide breakdown voltage.

4.4.9 GaAs MESFET

The GaAs MESFET model was derived from the model by H. Statz et al. at Raytheon. GaAs MESFET instances require the use of a model statement.

There have been some convergence problems with this model in the harmonic and spectral analyses that result from Cgs going to zero beyond pinchoff. The problem results when the gate is driven from an inductive source (such as in traveling wave amplifiers) and there is no other capacitance at the gate. To prevent this problem, it is suggested that Cgd not be set to zero and that side wall capacitance be added to the gate-source and gate-drain junctions. A good estimate for these capacitors is C = pi*epsilon*w/2 where w is the gate width in microns and epsilon = 0.116 fF/micron

ELEMENT DEFINITION:
Name drain gate source **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`area` Junction area in square meters
`region` DC operating region 0=off 1=on 2=rev 3=ohmic
MODEL DEFINITION:
model *ModelName* **gaas** <*parameter*=*value*> ...

MODEL PARAMETERS:
`nfet` N type GaAs MESFET
`nmf` N type GaAs MESFET
`pfet` P type GaAs MESFET
`pmf` P type GaAs MESFET
`vto` Pinch-off voltage
`beta` Transconductance parameter
`lambda` Channel length modulation parameter
`rd` Real (0.01): drain resistance.
`rs` Real (0.01): source resistance.
`is` Gate saturation current
`js` Gate saturation current

`cgs` Real (0.0): zero bias gate-source junction capacitance.
`cgd` real (0.0): zero bias gate-drain junction capacitance.
`pb` Gate junction potential
`fc` Forward-bias depletion capacitance threshold
`tnom` Parameter measurement temperature
`kf` Flicker-noise coefficient
`af` Flicker-noise exponent
`imax` Explosion current
`b` Doping tail extending parameter
`alpha` Saturation voltage parameter
`delta` Gate capacitance pinch-off transition width

4.4.10 Philips juncap diode model

The jucap diode model develope by Philips.

ELEMENT DEFINITION:
Name anode cathode **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`ab` Real (1.0e-12): diffusion area.
`area` Real (1.0e-12): diffusion area.
`ls` Real (1.0e-06): length of the sidewall of the diffusion area ab which is not under the gate.
`lg` Real (1.0e-06): length of the sidewall of the diffusion area ab which is under the gate.
`region` Boolean (0): supposed DC operating region: 0=off 1=on.
`m` Integer (1): the number of identical devices that are connected in parallel.
MODEL DEFINITION:
model *ModelName* **juncap** <*parameter*=*value*> ...

MODEL PARAMETERS:
`ntype` Boolean (false): the juncap is a ntype.
`ptype` Boolean (true): the juncap is a ptype.
`type` Tag (-): type of juncap. Admissible types are 'p' and 'n'.

`tnom` Real (): temperature at which the parameters have been determined.
`tref` Real (): temperature at which the parameters have been determined.
`tr` Real (): temperature at which the parameters have been determined.

`vr` VR
`dta` DTA
`jsgbr` JSGBR
`jsdbr` JSDBR
`jsgsr` JSGSR
`jsdsr` JSDSR
`jsggr` JSGGR
`jsdgr` JSDGR
`nb` NB
`ns` NS
`ng` NG
`vb` VB

`cjbr` CJBR
`cjsr` CJSR
`cjgr` CJGR
`vdbr` VDBR
`vdsr` VDSR
`vdgr` VDGR
`pb` PB
`ps` PS
`pg` PG

4.5 misc device

4.5.1 analog to digital converter

This element implements a pseudo digital to analog converter, that links the digital circuit to the electrical one.

ELEMENT DEFINITION:
Name pos neg **d2a** *<parameter=value> ...*

ELEMENT PARAMETERS:
`dignet` String (-): the hierarchy path inside the design that identifies the net to which the converter is connected. For example 'Mod.U1.e' specifies the net 'e' of instance 'U1' that in turn is instantiated in the 'Mod' root module.
`bits` list (-): the list is composed of two items, i.e. it has the form "bits=[b1,b0]", where 'b0' specifies the least significant bit of the digital net and 'b1' is the most significant one. In the convertion only these bits are considered.
`trtime` Real (-): the minimum allowed time window into which a variation of the analog output generated by the d2a conveter must happen.
`vl` Real(-): the voltage value that defines the low logic state.
`vh` Real(-): the voltage value that defines the high logic state.
`vx` Real(0.5*("vh"+"vl")): the voltage value that defines the 'x' logic state. This value must be less than "vh" and greater than "vl".
`rout` Real (1.0): the output port of the d2a is modeled as a driven current source with a resistor connected in parallel. This parameter sets the value of the resistor. Note that when the strength of the digital signal driving the d2a becomes 'Z', the conductance of this resistor is automatically set equal to 'gmin'.
`rz` Real (1/1e-12): the output resistance of the d2a when the digital is in the 'z' (high impedance) state. If this parameter is not given that default value equals the reciprocal of the <gground> conductance.

4.5.2 analog to digital converter

This element implements a pseudo analog to digital converter, that links the electrical circuit to the digital circuit.

ELEMENT DEFINITION:
Name pos neg **a2d** *<parameter=value> ...*

ELEMENT PARAMETERS:

`dignet` String (-): the hierarchy path inside the design that identifies the net to which the converter is connected. For example 'Mod.U1.e' specifies the net 'e' of instance 'U1' that is in turn instantiated in the 'Mod' root module.

`vl` Real(-): the voltage threshold that defines the low logic state. The analog voltage is coded in the corresponding binary value. All the bits of the digital coding are at the logic level 1 for analog values greater than <vh> and all the bits are at the logic level 0 for analog values less than <vl>. When the digital net is composed of only 1 bit the value of this parameter is set equal to that of the <vh> one. If the values of <vh> and <vl> parameters differ the average value is taken.

`vh` Real(-): the voltage threshold that defines the high logic state. In case of a digital net (variable), composed of more than 1 bit the analog voltage is coded in the corresponding binary value. All the bits of the digital coding are at the logic level 1 for analog values greater than <vh> and all the bits are at the logic level 0 for analog values less than <vl>. When the digital net is composed of only 1 bit the value of this parameter is set equal to that of the <vl> one. If the values of <vh> and <vl> parameters differ the average value is taken.

`vt` Real(-): the unique voltage threshold used to define the switching threshold of a 1 bit converter.

`trtime` Real (-): the minimum allowed time window into which a variation of the digital output generated by the a2d conveter must fall. During a time domain analysis, the integration time step is shortened such that any bit change of the coding of the digital net corresponding to a variation of the analog voltage falls inside a time window of length less than the value of this parameter.

4.5.3 Current probe

Current probe is a pseudo-element employed exclusively by the harmonic and the spectral analyses. It behaves as an independet current source at the specified harmonics and as a short circuit at the remaining harmonics of the spectrum.

ELEMENT DEFINITION:
Name n+ n- **iprobe** *<parameter=value> ...*

ELEMENT PARAMETERS:

`p1mag1, ...` Real (0.0): the magnutude of the current probe. In case of the <spectral> analysis this probe acts on the first fundamental and its harmonics.

`p1ph1, ...` Real (0.0): the phase of the current probe. In case of the <spectral> analysis this probe acts on the first fundamental and its harmonics.

`p1g1, ...` Real (0.0): the value of the conductance connected in parallel to the current probe. In case of the <spectral> analysis this probe acts on the first fundamental and its harmonics.

`freq2t` List [-]: the list of indices that defines the frequency of the second probe as a linear combination of the two fundamentals of the <spectral> analyses. It is composed of two items. The former is the index of the harmonic of the first fundamantal and the latter is the index of the harmonic of the second fundamental. For example 'freq2t=[1,-1]' defines the frequency (1*f1 -1*f2) in the <spectral> analysis.

`p2mag` Real (0.0): the magnutude of the second current probe in the <spectral> analysis.

`p2ph` Real (0.0): the phase of the second current probe in the <spectral> analysis.

`p2g` Real (0.0): the value of the conductance connected in parallel to the current probe acting on the second fundamental and its harmonics.

4.5.4 Microstrip Line

A microstrip line based on the equations of Hammerstad and Jensen. The model contains a the thickness correction to the width and frequency dependent permittivity and characteristic impedance. The dispersion equations are those of Kirschning and Jansen.

ELEMENT DEFINITION:
Name port1 port2 gnd **msline** *<parameter=value> ...*

ELEMENT PARAMETERS:
`l` Length in meters
`w` Width in meters
`h` Substrate height in meters
`t` Conductor thickness in meters
`eps` Substrate permittivity relative to a vacuum

4.5.5 Linear N Port

The characteristics of the N port versus frequency are determined by reading an S-parameter data file. S-parameters maybe entered in one of the following formats: real-imag, mag-deg, mag-rad, db-deg, db-rad. If the frequency scale factor is given, it is multiplied by the frequencies given in the data file to find the frequencies used by the simulator. Interpolation and extrapolation of the data is done using cubic splines on the data in polar form. A simple-minded algorithm is used to remove 2 pi jumps in the phase data. This requires that the frequency points at which the data is measured must be close enough to avoid an excessive number of jumps. Unfortunately, noisy phase data can confuse the algorithm and result in unnecessary warning messages. The file name is a quoted string. It is very unwise to rely on extrapolated data. The N port model supports also the definition of its electrical characteristic through the adoption of an external macromodel. The 'external' word means that it is a shared object run-time loaded by the simulator and then suitably executed. The electrical characteristic of the N port is descibed by the implicit equation:

G * v + Ai * i + R = 0

where G and Ai are a N x N matrices being N the port number, v is the vector of port branch voltages, i the vector of port branch currents and R is the N vector of known terms. The routine describing the macromodel is called with the following ordered sequence of arguments

<input> int : an index that uniquely identifies the instance of the N port. This index starts from 0 and it is incremented by 1 each time a new instance is found in the netlist; <input> int : number of ports of this N port; <input> double * : the array "v" of the port voltages; <input> double * : the array "i" of the port currents; <output> double **: the "G" matrix; <output> double **: the "Ai" matrix; <output> double * : the "R" vector of known terms; <input> double : the value of simulation time. It has meaning only in time domain analyses

The possible routine handling parameters of the macromodel is called with the arguments

<input> int : the index that uniquely identifies the instance of the N port. <input> char *: parameter name; <input> double: parameter value.

It must return 1 upon successful execution and 0 otherwise.

ELEMENT DEFINITION:
Name port1+ port1- <a lot of terminals> **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:
`func1, ...` The functions that describe the implicit characteristic of the NPORT. There must be one describing function for each port of the NPORT. Ports are formed by pairs of

terminals; each port is identified in the expression by the 'p' character followed by the port number. Ports are numbered by starting from 1. For example a linear resistor of 2 ohm can be decribed as 'Nport1 pos neg nport func1=v(p1) - 2*i(p1)'.

`sensedev` String (-): the name of the instance of the sensing device. The current through this device is used to control the NPORT. If the instance of the NPORT belongs to a subcircuit the sensing device is searched locally. This means that is a local and global devices with the same name exist the local one is chosen. There can be more than one of this parameter according to the number of input ports. This parameter applies when the NPORT characteristic is described by algebraic functions.

`gbsensedev` String (-): the global name of the instance of the sensing device. The current through this device is used to control the NPORT. Only global devices are considered; devices at the top-level of the netlist, i.e. those not belonging to subcircuits, are referred to as global. If a global and a local device exist, the global one is selected. If the global one does not exist, an error is displayed. There can be more than one of this parameter according to the number of input ports. This parameter applies when the NPORT characteristic is described by algebraic functions.

`sensen` String (-) / list(-): the list composed of two nodes forming a pair across which voltage is sensed. This voltage is used to control the NPORT. Pairs are identified by a label formed by the 'n' character followed by an integer numbering the pair. Pairs are numbered from integer 1 in increasing order as they are defined in the NPORT card. Intead of a list of two nodes, a string specifing 1 node can be given. In this case the pair is formed by the given node and ground.

MODEL DEFINITION:

model *ModelName* **nport** *<parameter=value>* ...

MODEL PARAMETERS:

`damping` Boolean (true): automatically tag some equations of macro-models in order to turn-on a damped version of the Newton method. In general this increses robustnes of analyses by avoiding drawbacks such as overflows, if for example exponential functions are used in the macro-model.

`file` String (-): S-parameter data file name.

`scale` Real (1.0): frequency scale factor.

`veriloga` String/list (-): the path/s of the file/s that must be compiled in order to implement an <veriloga> module.

`adms` String/list (-): the path/s of the file/s that must be compiled in order to implement an <veriloga> module.

`verilogapp` Boolean (false): the output of the <veriloga> pre-processor is saved in a file named as the module followed by the '.pp' suffix.

`verilogamodname` String (-): the name of the veriloga module linked to the <nport> model. If this parameter is omitted the module with the same name of the <nport> model is looked for.

`verilogaelectrical` Boolean (true): automatically add the <electrical> discipline if it is not specified in the veriloga file.

`verilogashadowvar` Boolean (true): a warning is given when a local variable shadows a previous one with same name.

`macro` Boolean (false): the device is described by an external macromodel loaded as a shared

executable module.

`module` String (-): the name (file path) of the shared executable implementing the macromodel.

`evaluate` String (-): the name of the routine in the shared executable that evaluates the macromodel.

`args` String (-): the name of the routine that handles the arguments passed to the macromodel.

`limiting` String (-): the name of the routine in the shared executable that does limiting of the electrical variables. Limiting should be an aid to convergence of the Newton algorithm when strong nonlinearities are involved.

4.5.6 Independent Resistive Source

A port is a resistive source used when calculating S-parameters. The port is tied between n+ and n-. R is the reference resistance. It must not be zero or negative and has a default value of 50 ohms. Num is the port number, if not given the number is assigned sequentially. The port is equivalent to a voltage source in series with a resistor. The DC voltage of the source is supplied using vdc, it specifies the DC voltage across the port when it is terminated in its reference impedance (in other words, vdc equals half the DC voltage supplied by the internal voltage source). In an AC analysis, mag and phase give the power delivered by the port when terminated with the reference resistance. Similarly, mag1, phase1, mag2, ... give the power delivered at each of the harmonics in a harmonic analysis and mag1f, phase1f, mag2f, and phase2f give the power at each of the fundamentals in a spectral analysis. Magnitudes are all given in dBm and phase is give in degrees. DC voltage is modifiable.

ELEMENT DEFINITION:
Name n+ n- **port** <*parameter=value*> ...

ELEMENT PARAMETERS:

`r` Reference resistance

`num` Port number

`vdc` DC voltage

`mag` Small signal power (dBm)

`phase` Small signal phase

`mag1` Power (dBm) of fundamental

`phase1` Phase of fundamental

`mag2` Power (dBm) of harmonic 2

`phase2` Phase of harmonic 2

`mag3` Power (dBm) of harmonic 3

`phase3` Phase of harmonic 3

`mag4` Power (dBm) of harmonic 4

`phase4` Phase of harmonic 4

`mag5` Power (dBm) of harmonic 5

`phase5` Phase of harmonic 5

`mag6` Power (dBm) of harmonic 6

`phase6` Phase of harmonic 6

`mag7` Power (dBm) of harmonic 7

`phase7` Phase of harmonic 7

`mag8` Power (dBm) of harmonic 8

`phase8` Phase of harmonic 8

`mag9` Power (dBm) of harmonic 9

`phase9` Phase of harmonic 9

`mag10` Power (dBm) of harmonic 10
`phase10` Phase of harmonic 10
`mag1t` Power (dBm) of first fundamental.
`phase1t` Phase of first fundamental.
`mag2t` Power (dBm) of second fundamental.
`phase2t` Phase of second fundamental.
`noise` Real (-): power spectral density of white noise (dBm) expressed as V^2/Hz. Note that if this parameter is given its value overwirtes the magnitude of the equivalent noise source derived from the given series resistance of the port. In other words the series resistor is considered as noiseless.

4.5.7 Poly Silicon Thin Film Transistor

Poly silicon thin film transistor (Poly-Si TFT) model has 3 electrical terminals. Poly-Si TFT model has the feature of binning. The binning equation is given by

P = P0 + Pl / Leff + Pw / Weff + Pp / (Leff * Weff)

Only the P0 parameters are listed and marked with BIN in description. Pl, Pw, and Pp are not shown but can be recognized. The names of Pl, Pw, and Pp are identical to that of P0 but with a prefix of pl, pw, and plw, respectively.

ELEMENT DEFINITION:
Name drain gate source **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:
`w` Channel width.
`l` Channel length.
`nrd` Drain squares.
`nrs` Source squares.
`m` Multiplicity factor (number of MOSFETs in parallel).
`region` Estimated operating region. Possible values are off, triode, sat, or subth.
`noisetype` Integer (5): types of generated noise; possible choices are:
0x0 : noiseless resistor; 0x1 : thermal noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.
`rth0` Thermal resistance.
`cth0` Thermal capacitance.
MODEL DEFINITION:
model *ModelName* **psitft** <*parameter*=*value*> ...

MODEL PARAMETERS:
`type` Transistor type. Possible values are n or p.
`vto` Threshold voltage at zero body bias (BIN).
`lambda` Real (0.048): channel length modulation parameter (BIN).
`tox` Gate oxide thickness.
`eta` Subthreshold ideality factor (BIN).
`etai` Alias to eta (BIN).
`asat` Proportionality constant of Vsat (BIN).
`alphasat` Alias to asat (BIN).
`delta` Transition width parameter (BIN).
`mus` Subthreshold mobility (BIN).
`mu0` High field mobility (BIN).

`muo` Alias to mu0 (BIN).
`mu1` Low field mobility parameter (BIN).
`mmu` Low field mobility exponent (BIN).
`m` Alias to mmu (BIN).
`vfb` Flat band voltage (BIN).
`dd` Vds field constant (BIN).
`dg` Vds field constant (BIN).
`blk` Leakage barrier lowering constant (BIN).
`clk` Leakage scaling constant (BIN).
`i0` Alias of clk (BIN).
`lkink` Kink effect constant (BIN).
`mkink` Kink effect exponent (BIN).
`mk` Alias of mkink (BIN).
`vkink` Kink effect voltage (BIN).
`rs` Source resistance.
`rd` Drain resistance.
`rsx` Resistance in series with Cgs.
`rdx` Resistance in series with Cgd.
`at` DIBL parameter 1 (BIN).
`bt` DIBL parameter 2 (BIN).
`eb` Barrier height of diode (BIN).
`i00` Reverse diode saturation current (BIN).
`ioo` Alias to i00 (BIN).
`etac0` Capacitance subthreshold ideality factor at zero drain bias (BIN).
`etac00` Capacitance subthreshold coefficient of drain bias (BIN).
`mc` Capacitance knee shape parameter (BIN).
`dvto` Temperature coefficient of VTO (BIN).
`dmu1` Temperature coefficient of MU1 (BIN).
`dasat` Temperature coefficient of ASAT (BIN).
`lasat` Coefficient of length dependence of ASAT (BIN).
`von` On-Voltage (BIN).
`cgso` Gate-source overlap capacitance.
`cgdo` Gate-drain overlap capacitance.
`vsigma` Above threshold DIBL parameter (BIN).
`vsigmat` Above threshold DIBL parameter (BIN).
`me` Long channel saturation transition parameter (BIN).
`ms` Alias to me (BIN).
`minme` Minimum value of me parameter..
`meta` ETA floating-body parameter (BIN).
`ls` Channel-length modulation coefficient 1 (BIN).
`vp` Channel-length modulation coefficient 2 (BIN).
`isubmod` Channel-length modulation model version.
`vmax` Carrier saturation velocity (BIN).
`theta` Mobility modulation coefficient (BIN).
`mss` Vdse transition parameter parameter (BIN).
`kss` Fractions of the channel resistance coefficient.
`rsh` Source/drain diffusion sheet resistance.
`capmod` Intrinsic charge model.
`zeroc` Zero gate-source (gate-drain) capacitance flag (Cgs=Cgd=0 if zeroc=1 and capmod=1).

`intdsnod` Intrinsic source and drain nodes usage flag.
`minr` Minimum source/drain resistance.
`tnom` Parameter measurement temperature.
`trise` Temperature rise from ambient.
`shmod` Self-heating selector.
`cth0` Self-heating thermal capacitance.
`rth0` Self-heating thermal resistance.
`wth0` Minimum width for thermal resistance calculation..
`tmax` Maximum device temperature above ambient.
`w` w
`l` l
`nrd` nrd
`nrs` nrs
`wmax` Maximum channel width for which the model is valid.
`wmin` Minimum channel width for which the model is valid.
`lmax` Maximum channel length for which the model is valid.
`lmin` Minimum channel length for which the model is valid.
`acm` Area calculation method.
`xl` Accounts for masking and etching effects.
`xw` Accounts for masking and etching effects.
`ld` Lateral diffusion into channel from source and drain diffusion.
`wd` Lateral diffusion into channel from bulk along width.
`lmlt` Length diffusion layer shrink reduction factor.
`wmlt` Width diffusion layer shrink reduction factor.
`ldif` Length of heavily doped diffusion adjacent to gate (BIN).
`hdif` Length of heavily doped diffusion from contact to lightly doped region.
`xj` Metallurgical junction depth.
`meto` Fringing field factor for gate to source and gate to drain overlap capacitance.
`rdc` Additional drain resistance due to contact resistance.
`rsc` Additional source resistance due to contact resistance.
`dvt` The difference between von and the threshold (BIN).
`vsi` Above threshold DIBL parameter (BIN).
`vst` Above threshold DIBL parameter (BIN).
`binunit` Bin parameter unit selector. 1 for microns and 2 for meters.
`binflag` Binflag=2 is to open xl/xw binning.

4.5.8 Time controlled switch

The switch is tied between n1 and n2 nodes and it is driven by the time sequense specified by means of the ton and toff time instants. There is no upper limit the the number of the ton and toff values. The resistance of the switch is set to <ron> when it is closed and to <roff> when open.

ELEMENT DEFINITION:
Name n1 n2 **ModelName** *<parameter=value> ...*

ELEMENT PARAMETERS:
`region` Integer (0): working status of the switch at the beginning of the time sequence: 1 = on 0 = off. For example if the status is on, the time sequence must start with a <toff> value that turns off the switch, followed by a <ton> value that turns on the switch and so on.
`ton` Real (-): time instant at which the switch is tuned on. There can be several of these time

instants.

`toff` Real (-): time instant at which the switch is tuned toff. There can be several of these time instants.

`period` Real (-): switching period; it is used only in time domain analyses.

`hindex` Integer (1): index of the harmonic at which the switch is operated. In spectral analysis the index is referred to the first fundamental.

MODEL DEFINITION:

model *ModelName* **switch** <*parameter*=*value*> ...

MODEL PARAMETERS:

`ron` Real (0.1m): resistance of the switch when it is on.

`roff` Real (1Meg): resistance of the switch when it is off.

`ts` Real (0.001): this parameter set the relative time interval with respect to the switching period (assumed normalised to 1) during which the switch commutes from the off to the on position or vice versa. Being a relative value it must be in the interval (0,1).

4.5.9 Transmission Line

A transmission line that includes dielectric and conductor loss. The conductor loss includes skin effect and does not assume the conductor is infinitely thick. All instance parameters are modifiable, Z0 is default. The currents into both ports are calculated. Currents are defined positive if they flow into the inner conductor, through the transmission line, and then out of the outer conductor.

Only the odd mode is modeled, so only the voltage difference across each port is important, not the absolute voltage of each terminal. To model both even and odd modes it is necessary to combine two lines. The line that models the odd mode will have both ports floating, the line modeling the even mode will have one node at each port grounded.

There are several different ways to specify the length of a line. First is to simply give the length and velocity (recall that velocity = 1/sqrt(relative dielectric constant)). The second is to give the reference frequency and the normalized length. The last is to give the delay in seconds.

There are three mutually exclusive ways to specify dielectric (shunt) loss; g, alphad, and qd. Both alphad and qd are low loss approximations. Similarly, there are three ways to specify conductor (series) loss; r, alphac, and qc. All imply a skin effect (loss that is proportional to the square root of frequency) and so require the use of fc. If dcr is given then the conductor is assumed to be of finite thickness and so the loss does not go to zero at DC, but rather approaches some constant value as frequency decreases below the corner frequency. It is also possible to specify loss by giving the DC resistance and the corner frequency. It is assumed that the loss from skin effect was measured at a frequency well above the corner frequency. This problem can be avoided by specifying loss with dcr and corner.

ELEMENT DEFINITION:

Name port1+ port1- port2+ port2- **ModelName** <*parameter*=*value*> ...

ELEMENT PARAMETERS:

`z0` Characteristic impedance

`td` Time delay

`f` Reference frequency

`nl` Normalized length in wavelengths at f

`vel` Propagation velocity normalized to c

`len` Physical length in meters

MODEL DEFINITION:

model *ModelName* **tline** <*parameter=value*> ...

MODEL PARAMETERS:

`f` Reference frequency
`vel` Propagation velocity normalized to c
`r` Series resistance per meter at fc
`g` Shunt conductance per meter
`corner` Skin depth corner frequency
`alphac` Conductor loss at fc
`alphad` Dielectric loss
`dcr` DC series resistance per meter
`qc` Conductor loss quality factor
`qd` Dielectric loss quality factor
`fc` Conductor loss measurement frequency
`fd` Dielectric loss measurement frequency

4.5.10 Voltage controlled switch

The voltage controlled switch is tied between terminals n1 and n2 and it is driven by the voltage across the n3 and n4 terminals. The switch resistance varies from the off value (switch is off) to the on one (switch is on). This variation is linear with respect to the controlling voltage in the [voff,von] range. The noise model refers to the varying output resistance of the voltage controlled switch. Input resistance in noiseless.

ELEMENT DEFINITION:
Name n1 n2 sense+ sense- **ModelName** <*parameter=value*> ...

ELEMENT PARAMETERS:

`noisetype` Integer (5): types of generated noise; possible choices are:
0x0 : noiseless voltage switch; 0x1 : thermal noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

MODEL DEFINITION:

model *ModelName* **vswitch** <*parameter=value*> ...

MODEL PARAMETERS:

`ron` Real (1.0e-3): on resistance. The switch is considered on when the controlling voltage is greater than von if von > voff and vice versa when the controlling voltage is less than von if von < voff.
`roff` Real (1.0e6): off resistance. The switch is considered off when the controlling voltage is less than voff if von > voff and vice versa when the controlling voltage is greter than voff if von < voff.
`rin` Real (gmin): input resistance of the controlling port.
`von` Real (1.0): on voltage threshold.
`voff` Real (-1.0): off voltage threshold.

`af` Real (1.0): flicker noise exponent.
`kf` Real (0.0): flicker noise coefficient.

4.6 Basic analyses

4.6.1 DC Analysis

The DC analysis computes the operating point of linear and nonlinear circuits. Furthermore it is possible to sweep one or more circuit parameters and compute the circuit operating point at each value of the parameters. Computation is done through the Newton iterative method. The DC analysis implements some continuation methods such as gmin-stepping, source-stepping, ggroung-stepping and pseudo-transient. These methods should improve robustness and are called only when the conventional DC analysis failes. The DC analysis supports also thermal networks, in other words it is possible to carry out an electro-thermal simulation where the self-heating of devices is considered and the solution of the circuit accounts for the present values of device temperatures. When dealing with large circuits, they are partitioned to increase efficiency and robustness. The DC analysis can look for possible multiple solutions. The implemented method has been described in

G. Storti-Gajani, A. Brambilla, A. Premoli, "Numerical Determination of Possible Multiple DC Solutions of Nonlinear Circuits", IEEE Tranaction on CAS-I, Vol. 55, No. 4, May 2008, pp. 1074-1083.

ANALYSIS DEFINITION:

Name **dc** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (3): degree of annotation. Values from 1 to 7 turn on display of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08 : at each iteration of the Newton method, some information about the partial solution that maximally violates accuracy thresholds; is displayed;
0x10 : at each iteration of the Newton method, values of partial solutions are displayed;
0x20 : at each iteration of the Newton method, information about current values of partial solution, their increments and values of RHS is displayed.
0x40 : print MNA matrix, RHS vector and the DeltaSolution vector at each iteration of the Newton method.
0x80 : dump MNA matrix in the "analysis_name.m" file at each iteration of the Newton method.
0x100: the histogram of the size of sub-circuits, loops and branches handled or identified by the circuit partitioner is displayed. An example of the format displayed is
Loops : 2 ***** 3467 8 where '2' refers to the loop number, '*****' is the histogram, '3467' is the the number of circuit nodes belonging to the loop and '8' is the cardinality of the cut that opens the loop.
Branches : 2176 ***** 77 L 33 where '2176' refers to the circuit node number, '*****' is the histogram, '77' is the number of branches, 'L' tells that the node belongs to s loop, another possible character is 'S' telling that the node belongs to a sub-circuit and finally '33' shows the number that identifies the loop or sub-circuit, respectively. Note that sizes less than a lower threshold are not displayed.

`maxiters` Integer (100): maximum number of iterations.

`print` Boolean (false) or list(-): print operating point or a subset of node voltages and branch currents as specified in the given list. Regular expressions can be used to specify the items of the list.

`printnodes` Boolen (false): print external/internal node names-index map. This parameter serves mainly for debug purposes.

`instance` String (-): device whose parameter value is to be swept.

`opt` boolean (false): tell that a simulator options has to be swept.
`param` String (-): name of parameter to be swept.
`center` Real (-): center of sweep
`dec` Integer (-): points per decade
`lin` Integer (50.0): number of points, linear sweep
`log` Integer (50.0): number of points, log sweep
`span` Real (0.0): sweep limit span
`start` Real (0.0): start sweep limit
`step` Real (-): step size, linear sweep
`stop` Real (-): stop sweep limit
`resetsweep` Boolean (true): set back all sweeping parameters at the values they had at the beginning of the sweeping analysis. If set to false parameters are left at values assumed at the end of the sweeping analysis.
`checklc` Boolean (false): check and reduce large conductance trees.
`lclocal` Boolean (false): use local algorithm for large conductance reduction.
`safetydigits` Integer (0): how many more digits of the ALU should be preserved by the large conductance transformation.
`restart` Boolean (true): restart the DC analysis; this means that results of the previous analysis are not used as initial guess
`saman` Boolean (false): turn on the Newton-Samanskii method. This method tries to not update and factorise the Jacobian matrix at each iteration of the Newton method. The Jacobian, that was factorised at a step of the Newton algorithm, is keep for several following iterations of Newton algorithm. The norm of the RHS vector is monitored and if it does not decrease at each iteration of the Newton-Samanskii method, the Jacobian is re-factorised and the Newton-Samanskii method restarted.
`thermalnet` Boolean (false): use the thermal network.
`keepmatrixorder` Boolean (true): when true the algorithm that LU factorise the Jacobian matrix at each iteration of the Newton method tries to keep the same matrix order. In general when the Jacobian matrix is LU factorised the algorithm tries to limit the number of generated fill-ins (extra entries in the sparse Jacobian matrix) and the growth of the round-off error. This is accomplished by reordering the rows and the columns of the Jacobian matrix. Reordering requires the identification of the pivoting entries and submatrix reduction by meas of row/column linear combinations. This is an extremely cpu time consuming task, that often takes a very large portion of the total simulation time, mainly in large circuits. Experience shows that if the Jacobian matrix is LU factorised by keeping a fixed reordering schema the total simulation time reduces of about 6 times. The main drawback is that the growth of the round-off error is badly controlled and can lead to a failure of the Newton method. Once more experience shows that the Newton algorithm by its intrinsic iterative nature is almost always able to dump the effects of the round-off errors, but still there can be circuits yielding a very bad conditioned Jacobian matrix that can cause convergence problems.
`stab` Integer (0): the stability analysis of the computed equilibrium point is performed. The pole with the largest real value is determined; if it is positive, the circuit is unstable. Two different solvers can be chosen:
0: no stability analysis is performed;
1: a direct solver is employed;
2: an iterative solver is employed.
`nport` Boolean (false): consider devices, mainly mosfets, as nports, i.e. their internal nodes are dropped during the solution of the circuit. The solution at internal nodes are computed

through the Newton iterative method that is applied locally to each device.

`partitioner` Integer (0): this option turns on the circuit partitioner. Allowed values are:
0 : the circuit partitioner is kept-off;
1 : the circuit partitioner is turned-on.
The circuit partitioner considers unidirectional elements such as mosfets and partitions the circuit into sub-circuits. The solution of the sub-circuits is organised in a directed graph with no cycles (flow graph). The flow graph is traversed (this means that the sub-circuits are solved) by starting from the input nodes (those that have only outgoing edges) to the ending nodes (those that have only ingoing edges). If circuit partitioning option is not specified, it is automatically set equal to 1, when the total number of mosfets (of various types) plus bjts exceeds a predefined upper limit.

`supernode` Boolean (true): introduce custsets formed by independent voltage sources. These cutsets are known as 'super-nodes'. Super-nodes avoid to add the current through the independent voltage source among the unkowns of the problem and thus reduce the size of the problem increasing efficiency.

`subeqns` Integer (1000): number of equation to feed each thread circuit partitioning.

`tarjan` Boolean (false): turn on/off the EXPERIMENTAL Tarjan/DulmageMendelsohn matrix block triangularization algorithm.

`tarjanthresh` real/integer (-1): set Tarjan threshold value (if positive real) or threshold policy mode (if negative integer). default value is -1 and corresponds to standard no threshold Tarjan

`splitloop` Boolean (false): try to split too large loops of subcircuits generated during circuit partitioning. Splitting is performed by 'cutting' nodes and intruducing artificial voltage sources.

`parsolver` Boolean (flase): turn on/off the parallel solver by using multiple threads.

`pardevices` Boolean (flase): turn on/off the parallel evaluation of device models by using multiple threads.

`hierckt` Boolean (flase): turn on/off the solution of the hierarchical version of the circuit. In a conventional analysis, a circuit hierarchically organised through subcircuits and instances of subcircuit, is first flattened, i.e. a flat version of the netlist is generated and then the circuit is solved. If this flag is true the circuit is not flattened and is solved on a subcircuit bases.

`maxabsdeltav` Real(-): maximum allowed change of node voltages at each iteration of the Newton algorithm.

`maxreldeltav` Real (0.999): maximum allowed relative change of node voltages at each iteration of the Newton algorithm. The relative value refers to the largest node voltage by the Newton method. If the relative voltage variation exceeds the maximum allowed value the Newton step is suitably shrink to meet this limit. The Newton direction is preserved.

`maxdeltat` Real (-): max change in temperature allowed per iteration. This parameter is effective only during an electro-thermal analysis

`maxdevtemp` Real (1300.0 Celsius): maximum allowed device temperature

`vmax` Real (-): maximum voltage allowed during iterations of the Newton algorithm

`vmin` Real (-): minimum voltage allowed during iterations of the Newton algorithm.

`vmaxguess` Real (0.0): maximum voltage allowed during evaluation of the initial guess of the Newton algorithm. If a node voltage is above this value it is set to the vmax value. By setting vmax to 0 and vmin to 0 the Newton algorithm starts with a 0 initial guess and thus the evaluation of the starting value for the Newton algorithm has no effect

`vminguess` Real (0.0): minimum voltage allowed during evaluation of the initial guess of the Newton algorithm. If a node voltage is below this value it is set to the vmin value

`ireltol` Real (1.0e-6): relative current convergence criterion

`iabstol` Real (1.0e-12): absolute current convergence criterion

`vreltol` Real (1.0e-6): relative voltage convergence criterion.

`vabstol` Real (1.0e-6): absolute voltage convergence criterion.

`ic` List (-): it is a list specifying the initial value of node voltages and branch currents employed as a guess for the Newton iterative method. These values override those automatically computed by the <dc> analysis as initial guess. A dc analysis is initially done where the ic conditions "force" the Newton method. When convergence has been achieved the Newton method runs "free" to find the operating point of the circuit with no more constraints due to ic conditions. It is not ensured that the specified values are met at convergence. The list has the format <ic=[node,value,branch,value,...]>. These initial conditions are "implemented" as Norton equivalents connected to the forced equations where the parallel resistor has a conductance equal to the product of the <icforce> parameter by the diagonal entry of the MNA matrix corresponding to the forced equation. The magnitude of the independent current sources are so that to initially force the given initial conditions. The Norton equivalents are swept out during the dc analysis.

`icforce` Real (1.0): the initial value of the coefficient employed to force the initial conditions for the Newton algorithm. It has meaning/effect only if the <ic> parameter is specified.

`damping` Boolean (false): apply a version of the damped Newton method.

`srcstepping` Boolean (true): turn on/off the source stepping continuation algorithm.

`gminstepping` Boolean (true): turn on/off the gmin stepping continuation algorithm.

`ggroundstepping` Boolean (true): turn on/off the gground stepping continuation algorithm.

`pseudotransient` Boolean (true): turn on/off the speudo-transient continuation algorithm.

`initgground` Real (-): this parameter has effect only when the pseudo-transient continuation method is started. The initial value of the parasitic conductance used to emulate the pseudo-transient is set to the value specified by this parameter instead of starting from 100 times the value of the <gground> parameter and increasing it till the Newton algorithm begins to converge.

`initgmin` Real (-): this parameter has effect only when the gmin continuation method is started. The initial value of the parasitic conductance is set to the value specified by this parameter instead of starting from 100 times the value of the <gmin> parameter and increasing it till the Newton algorithm begins to converge.

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial guess to the Newton algorithm.

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This solution is used as initial guess to the Newton algorithm.

`mem` List (-): results that have to be saved in memory. These results can be later accessed from the body of the <control> analysis through the <get> function. Note that the saving of results in memory can require and occupy a large amount of memory.

`multiple` Boolean (false): multiple solutions of the electrical circuit are possibly searched. Note that the adopted approach does not ensure to find all the solutions of the circuit. To find possible solutions some peculiar independent voltage sources are inserted in the circuit. The magnitude of these sources is swept; stat, stop and step values are automatically determined. The user can modify these values by acting as in sweep analysis (see the mealing of the related parameters).

4.6.2 Small Signal Analysis

This analysis linearizes the circuit about the DC operating point and performs a small signal analysis. If desired, the analysis will be repeated while sweeping some independent variable. The variable may be either frequency or some device instance parameter. Not all device instance parameters are modifiable and some of those will change the DC operating point when modified. If changing a parameter affects the DC operating point, the operating point is recomputed at each step. The modified parameter is reset to its original value after the analysis. This analysis can determine also the poles and zeros of the linearised circuit. In particular to compute zeros a transfer function must be defined by specifying the input source and the output voltage or current.
ANALYSIS DEFINITION:
Name **ac** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:
`annotate` Integer (default is global annotate level): degree of annotation. Values from 1 to 7 turn on display of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08 : print MNA matrix; 0x10 : dump MNA matrix in the "analysis_name.m" file.
`freq` Real (-): the frequency value when a device value parameter is swept or the frequency value in single frequency analysis. This parameter must not be specified contemporary to the frequency sweeping command.
`print` Boolean (false): print the result in single frequency analysis.
`start` Start sweep limit
`stop` Stop sweep limit
`center` Center of sweep
`span` Sweep limit span
`step` Step size, linear sweep
`lin` Number of points, linear sweep
`dec` Points per decade
`log` Number of points, log sweep
`sweep` Device whose parameter value is to be swept
`param` Name of parameter to be swept

4.6.3 Time domain analysis

The analysis computes the time domain behavior of linear and nonlinear circuits. The initial conditions are computed automatically through a DC analysis or can be specified by the user. In this case the user gives the initial voltage across capacitors and current through inductors. Time domain integration employs different types of linear multi-step methods, such as backward Euler, trapezoidal and Gear methods. There are two different mechanisms to control the integration time step. Waveforms made available to control analysis are:

1) Node voltages and branch currents computed by transient analysis, time samples are stores in the 'time' vector. Waveforms can be accessed through the command 'get("TranName.node")'.

2) Spectra of time domain node voltage and branch current waveforms, frequency samples are stored in the 'freq' vector. The spectrum of a node voltage and branch current is named as the electrical variable followed by the ':spectrum' postfix. Therefore spectra can be retrieved through the command 'get("TranName.node:spectrum")'.

3) When the parameter to compute the working period of a periodic circuit is turned on, these other results are available: 'period' the computed working period of the circuit; 'tstart' the starting time instance of a computed period; 'tstop' the last time instant of the computed period;

'samples' the index identifying the working period (x-axis of these results).

4) The monodromy matrix of the circuit can be retrieved through the command 'get("TranName.monodromy")'. The vector that maps the column index of the nonodromy matrix to the corresponsing MNA equation can be retrived through the command 'get("TranName.eqmap")'.

5) Results by time domain noise analysis can be retrived through the command 'get("TranName.node:noise")'. Results are computed on a regular time point grid. Time instants can be retrieved through the command 'get("TranName.noisetime")'.

ANALYSIS DEFINITION:
Name **tran** <*parameter=value*> ...

ANALYSIS PARAMETERS:

`annotate` Degree of annotation. It is an integer number values from 1 to 7 turn on displaying of generic informations about the analysis run. Values above 7 are bitwise and turn on specific printouts. In particular: 0x08: print some information about the convergence of the Newton method;
0x10: print partial solutions at each iteration of the Newton method;
0x20: print MNA matrix and RHS vector at each iteration of the Newton method.

`checkstrange` Boolean (1): check if there are some 'strange behaviours' of devices. For example, it is checked if bulk-drain and bulk-source parasitic diodes of mosfets are forward biased. This check slows down a bit the simulation and can generate a large number of warnings.

`safetydigits` Integer (0): how many more digits of the ALU should be preserved by the large conductace transformation.

`maxdevtemp` Maximum allowed device temperature (Celsius)

`maxiter` Integer (10): maximum allowed number of Newton iterations at each time step of the time domain analysis. The Newton algorithm is considered as non converging when the iteration number goes above the 'maxiter' value. In this case the integration time step is shortened and the Newton algorithm is restarted.

`saman` Boolean (false): turn on/off the Newton-Samanski method.

`maxinititer` Integer (100): maximum allowed number of Newton iterations during the determination of the initial conditions from which the transient analysis is started.

`maxtminiter` integer (100): maximum allowed number of Newton iterations when the integration time step length is equal to the its minimum allowed value, specified through the 'tmin' parameter.

`printnodes` Boolen (false): print external/internal node names-index map. This parameter serves mainly for debug purposes.

`savelist` It defines a list on nodes or branch currents that will be saved into the output file during simulation. Other computed electrical variables are not saved

`devcurr` Boolean (false): the currents of devices, for example the drain current of mosfets, the anode-cathode current of diodes and currents through capacitors are stored in the simulation output file.

`thermalnet` Use the thermal network

`partitioner` Boolean (false): this option turns on/off the circuit partitioner.

`supernode` Boolean (false): introduce custsets formed by independent voltage sources. These cutsets are known as 'super-nodes'. Super-nodes avoid to add the current through the independent voltage source among the unkowns of the problem and thus reduce the size of the problem increasing efficiency.

`subeqns` Integer ((1000)): number of equation to feed each thread circuit partitioning.

`threads` Boolean (flase): turn on/off the parallel analysis by using multiple threads.

`method` Select the integration method:
1 trapezoidal (default); 2 Gear method of various order.

`order` Set the order of the Gear integration method, default value is 2. It has no meaning if other integration methods are employed

`acntrl` Integer (2): select the accuracy control method:
1 relative energy balance; this method computes the electrical energy variation of each capacitor and inductor along each integration interval and the corresponding electrical work delivered by the resistive one-port elements to which they are connected. The electrical work and energy are compared and the integration time step is chosen in order to keep their difference below the user defined threshold;
2 absolute energy balance. This method is identical to the previous one, but the difference between the electrical work and energy is compared to the total energy stored into the corresponding capacitor/inductor. The integration time step is chosen to keep variation with respect to total stored energy below the user defined threshold. In general this method gives less accurate results with respect to the previous one, but the analysis runs faster.

`ntoltype` Integer (2): this option specifies the tolerance check adopted at each time point of the analysis to control if the Newton method has reached convergence. Checks are done both on the variation of variables computed by the Newton method and on the entries of the right hand side vector. Define as 'dX(n)' the variation of the 'n' electrical variable, as 'reltol' and 'abstol' the relative and absolute maximum allowed variations, respectively. The n-th variable has converged if '| dX(n) | < reltol * Y(n) + abstol'. The 'ntoltype' parameter defines three different ways to compute 'Y(n)', number as 1, 2 and 3. Their meanings are listed in the following:
1 : global. 'Y(n)' is defined as the maximum over the time interval, till the current time point, of the maximum value assumed by the 'X(n)' electrical variables, where 'n' varies from 1 to N, being N the number of circuit variables. Note that in this case 'Y(n)' is unique value adopted for all the 'dX(n)' variable variations.
2 : local. 'Y(n)' is defined as the maximum absolute value of 'X(n)' in the time interval till the current time point. Note that in this case 'Y(n)' is different for each electrical variable.
3 : point local. 'Y(n) = | X(n) |'.
The convergence check tightens as 'ntoltype' is increased. Default value of this parameter is 2 (local).

`keepmatrixorder` Boolean (1): when true the algorithm that LU factorise the Jacobian matrix at each iteration of the Newton method tries to keep the same matrix order. In general when the Jacobian matrix is LU factorised the algorithm tries to limit the number of generated fill-ins (extra extried in the sparse Jacobian matrix) and the grouth of the round-off error. This is accomplished by reordering the rows and the column of the Jacobian matrix. Reordering requires the identification of the pivoting entries and submatrix reduction by meas of row/column linear combinations. This is an extremly cpu time consuming task, that often takes a very large portion of the total simulation time, mainly in large circuits. Experience shows that if the Jacobian matrix is LU factorised by keeping a fixed reodering schema the total simulation time reduces of about 6 times. The main drawback is that the grouth of the round-off error is badly controlled and can lead to a failure of the Newton method. Once more experience shows that the Newton algorithm by its intrinsic iterative nature is almost always able to dump the effects of the round-off errors, but still there can be circuit yielding to a very bad conditioned Jacobian matrix that can cause convergence problems.

`skipdc` Boolean (false): the initial DC analysis performed to determine the initial conditions is skipped. This can help when dealing with critical circuits that cause failures of the DC

analysis. In this case voltages across capacitors and current through inductors are assumed null.

`uic` Integer (0): use given initial conditions of inductors and capacitors. Initial conditions are considered in two different ways according to the value of this paramenter; in particular:
1: it is assumed that all the capacitors and inductors have suitable initial conditions. This means that if they are not specified by the user, they are assumed equal to zero. Note that this initial condition determination fails when there are loops of voltages source and capacitors even though consistent initial conditions are given, since the loop current is undertermined. The same applies to inductor cut-sets.
2: it is assumed that all the capacitors and inductors whose initial conditions have not been specified are open circuit and short circuit, respectively.

`restart` Boolean (true): restart the analysis, that is, do not use results computed by a previous transient, envelope or shooting analysis as initial condition for the current transient analysis.

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial condition from which the transient algorithm is started.

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This solution is used as initial condition to the transient algorithm.

`checklc` Boolean (false): find, check and reduce trees of large conductances. This can be an aid to convergence of the integration algorithm, since large conductance reduction prevents growing of round-off errors.

`lclocal` Boolean (false): use local algorithm for large conductance reduction.

`tmin` Real (-): minimum allowed time step length. The value of this parameter is automatically determined by the simulator if it is not specified by the user.

`tmax` Real (-) or List (-): maximum allowed time step length. If a list is given it has the format '[t1:t2,ts1,...]. The value of this parameter is automatically determined by the simulator if it is not specified by the user.

`tinc` Real ((1.1): lengthening factor of the time step at convergence.

`timepoints` List (-): the list containing the time instants that should be employed by the simulator during the analysis. It is not ensured that these time instants are effectively met. However during the analysis these time instants are considered and the integration time step selection algorithm tries to meet these points.

`savetime` Real (-): results are saved in the output file when simulation time is greater than the value of this parameter. This parameter can be useful to reduce the amount of results saved in the output file.

`tstart` Real(-): time instance at which the transient analysis begins.

`stop` Time stop or end of the time domain analysis

`tstop` Time stop or end of the time domain analysis

`trabstol` Real (1.0e-12): absolute convergence tolerance for capacitor voltages and inductor currents.

`trreltol` Real (1.0e-3): relative convergence tolerance for capacitor voltages and inductor currents

`vabstol` Absolute voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis (default 1.0e-6)

`vreltol` Relative voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis

`iabstol` Absolute current tolerance employed by the Newton algorithm at each time point of

the time domain analysis (default 1.0e-12)

`ireltol` Relative current tolerance employed by the Newton algorithm at each time point of the time domain analysis

`opiabstol` Absolute current tolerance for the evaluation of the initial conditions.

`opireltol` Relative current tolerance for the evaluation of the initial conditions.

`opvabstol` Absolute voltage tolerance for the evaluation of the initial conditions

`opvreltol` Relative voltage tolerance for the evaluation of the initial conditions

`maxdeltav` Real (-): the maximum allowed variation of node voltage allowed during the iterations of the Newton algorithm. This should aid convergence and avoid possible overflow.

`vmax` Real (-): maximum voltage allowed during iterations of the Newton algorithm

`vmin` Real (-): minimum voltage allowed during iterations of the Newton algorithm.

`damping` Boolean (false): apply a version of the damped Newton method.

`ggroundstepping` Boolean (true): turn on/off the gground-stepping continuation algorithm. It acts as 'fall-back' method only when the conventional transient analysis fails due to singular MNA matrix.

`gminstepping` Boolean (true): turn on/off the gmin-stepping continuation algorithm. It acts as 'fall-back' method only when the conventional transient analysis could fail.

`pseudotransient` Boolean (true): turn on/off the pseudo-transient continuation algorithm. It acts as 'fall-back' method only when the conventional transient analysis could fail.

`cmin` Boolean (true): if true parasitic capacitors of suitable values are inserted between each circuit node and ground when the simulator is in trouble to compute the solution. This can aid integration algorithms. However attention must be payed since insertion of these capacitors can yield some circuits to oscillate or even to become unstable!

`skip` Integer (1): skip time point (advance in time) in case of unrecoverable convergence failures of the Newton algorithm. It helps to continue and successfully terminate the simulation even though accuracy can be locally compromised. Possible choises of this parameter are:
0: no skipping is applyied in case there is no convergence of the Newton algorithms;
1: skipping is applyed but only if there is convergence of the Newton algorithm in the new time point, determined after the forward time skipping.
2: forward time skipping is applyed anyway, even though there is no convergence of the Newton algorithm in the new time point.

`ic` List (-): specifies the initial value of node voltages and branch currents employed in the transient analysis. These values override those automatically computed by the <dc> analysis as initial guess. A dc analysis is initially done where the ic conditions "force" the Newton method. It is not ensured that the specified values are met at convergence even though the result should be close to them. The list has the format <ic=[node,value,branch,value,...]>. For circuit nodes, these initial conditions are "implemented" as Norton equivalents connected to the forced node where the parallel resistor has a conductance equal to the product of the <icforce> parameter by the diagonal entry of the MNA matrix corresponding to the forced equation. The magnitude of the independent current sources are so that the product of their currents by the parallel resistors is equal to the given initial conditions.

`icforce` Real (1.0): the initial value of the coefficient employed to force the initial conditions for the Newton algorithm. It has meaning/effect only if the <ic> parameter is specified.

`fft` Boolean (false): turns FFT analysis on/off.

`fftfund` Real (-): the frequency of the fundamental. A fraction of the reciprocal of this value (time period) is employed to sample waveforms. For more details, see the <"fftperiod"> parameter.

`fftperiod` The period along which waveforms are sampled and FFT is performed. The

sampling interval should begin <fftperiod> before the <tstop> value. If the <fftestper> parameter is set true, the <"fftperiod"> value is automatically adjusted within a maximum value of +40

`fftestper` Boolean (false): estimation of the period of waveforms along which the FFT transformation is applied. If this parameter is true it is not mandatory to specify an accurate value of <fftperiod> since it is automatically adjusted within a maximum relative value of +40

`fftharms` Real (4): the number of harmonics computed by FFT.

`fftsamples` The number of sampling points in the FFT analysis. This parameters is effective only if it specifies a number of samples greater than '4 * <ffthars> + 1', which represents the minimum allowed number of employed samples.

`fi` Boolean (false): turn on and off the computation of the Fourier integrals, i.e. the computation of the components of the waveform spectra.

`fifund` Real (-): the frequency of the fundamental. A fraction of the reciprocal of this value (time period) is employed to sample waveforms. For more details, see the <"fiperiod"> parameter.

`fiperiod` The period along which Fourier integrals of the computed wavefors are performed. The interval should begin <"fiperiod"> before the <"tstop"> value. If the <fiestper> parameter is set true, the <"fiperiod"> value is automatically adjusted within a maximum value of +40

`fiestper` Boolean (false): estimate the period of waveforms to which Fourrier integrals are applied. If this parameter is true it is not mandatory to specify an accurate value of <ifperiod> since it is automatically adjusted within a maximum relative value of +40

`fifreq` List (-): list of frequecies at which the Fourier integrals are computed.

`findperiod` Boolean (false): find period values during the tran analysis. This can be used to monitor synchronization transients. Unconsistent results are possibly found if a nonperiodic dynamics is present. Period doubling may also cause problems.

`monodromy` Boolean (false): compute the monodromy matrix of the circuit.

`printmo` Boolean (false): print the monodromy matrix of the circuit.

`jump` Boolean (false): activate finding of discontinuities in the vector field. This option has meaning only when the fundamental matrix is computed. Saltation matrices are exploited to construct a correct fudamental matrix of circuits characterised by discontinuities in the vector field. Discontinuities in the vector field can be also introduced by sharp varying functions modeling circuit elements, such as for example the 'y=tanh(a*x)' function with the 'a' gain too large. The solver automatically determines possible discontinuities due to numerical problems introduced by too sharp nonlinear functions and automatically introduce 'saltation' matrices. The solver introduces also 'saltation' matrices due to dscontinuities caused by commutations of pseudo analog-to-digital and digital-to-analog converters during simulations of mixed digital/analog circuits and due to behavioural elements tagged as digital.

`lyapunov` Boolean (false): compute the Lyapunov exponents and the Kaplan-York dimension of the tran circuit trajectory. It is possible to chose different policies concerning the time instant at which monodromy is reset and QR computed: Time based (after a given time) or Sample based (after a fixed number of time points). Sample based is the default behaviour if parameter lyapstep is not specified.

`lyapstep` Real (-): Force Time Step for the computation of Lyapunov exponents (and resetting monodromy to identity).

`lyapsamp` Integer (100): Number of time samples that force computation of QR and resetting monodromy matrix.

`noisefmax` Real (-): maximum frequency considered in simulating time domain noise effects. The time domain noise analysis is automatically turned on by specifying this parameter.

`seed` Integer (-): the seed of the random number generators used in the time domain noise analysis. If not specified the seed is taken from the system clock.

4.7 Simulator control pseudo-analyses

4.7.1 Set Pan Options

This statement sets or changes various program control options. The options may be set in any order and once set retain their value until reset.

ANALYSIS DEFINITION:

Name **options** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`gmin` Real (1e-12): conducatnce of hidden resistors in general employed in equivalent models of nonlinear devices (for example diodes, mosfets, bjt) to improve relinability of models and convergence properties of numerical algorithms.

`gmax` Real (100): maximum value that can be assumed by <gmin> during gmin-stepping continuation method.

`gground` Real (1e-12): conductance of hidden resistors that connect each circuit node to ground. These resistors should aid convergence of numerical algorithms. If not given its value is assumed equal to that of the <gmin> parameter.

`maxgground` Real (100): maximum value that can be assumed by <gground> during gground-stepping continuation method.

`maxoscillations` Real (20): maximum allowed number of oscillations durinrg pseudo-transient continuation method.

`mindevres` Real (1.0e-3): the minimum allowed value that the possible resistors connected in series to the internal device nodes can assume. For example a diode has an internal resistor that models resistance of contancts and silicon that do not constitute the junction. When a resistor internal to a device model has a value less than that specified by this parameter, it is not considered and removed from the circuit.

`pivrelthresh` Real (0.001): this parameter determines what the pivot threshold will be. It should be between zero and one. If it is one then the pivoting method becomes complete pivoting, which is very slow and tends to fill up the matrix. If it is set close to zero the pivoting method becomes strict Markowitz with no threshold. The pivot threshold is used to eliminate pivot candidates that would cause excessive element growth if they were used. Element growth is the cause of roundoff error. Element growth occurs even in well-conditioned matrices. Setting <pivrelthresh> large will reduce element growth and roundoff error, but setting it too large will cause execution time to be excessive and will result in a large number of fill-ins. If this occurs, accuracy can actually be degraded because of the large number of operations required on the matrix due to the large number of fill-ins. A good value seems to be 0.001. This value should be increased and the matrix resolved if growth is found to be excessive. Changing the pivot threshold does not improve performance on matrices where growth is low, as is often the case with ill-conditioned matrices.

`pivabsthresh` Real (0.0): the absolute magnitude a matrix element must have to be considered as a pivot candidate. This number should be set significantly smaller than the smallest diagonal element that is expected to be placed in the matrix. This element has a value equal to the <gground> parameter. If there is no reasonable prediction for the lower bound

on these elements, then <pivabsthresh> is set equal to the square of <gground>. The <pivabsthresh> is used to reduce the possibility of choosing as a pivot an element that has suffered heavy cancellation and as a result mainly consists of roundoff errors.

`pivcaching` Boolean (true): when true the chaching mechanism to gather pivots during the LU factorization of the MNA matrix is adopted. Pivot chaching augments the effeciency in LU factorizing the matrix. In general the chaching mechanism does not worsen the conditioning of the MNA matrix, but in extreme cases this can possibly happen. In this case it is suggested to turn off the pivot chaching.

`annotate` Integer (-): set annotation level of simulator. Verbosity is expressed through an exadecimal number, higher bit levels augment annotation (bit enumeration starts from 0); the meaning of each bit is the following:
bit 1 : turns on default simulator annotation; bits 2-3: narrate parser actions; bit 4 : print some values of some device instances. bit 5 : print some info about digital nets.

`warn` Boolean (true): give warning messages.

`maxwarns` Integer (10): maximum number of given warnings. The other ones are omitted; in this case the total number of warnings (given and not given) is displayed at the end of the analysis.

`maxnotices` Integer (10): maximum number of given notices. The other ones are omitted; in this case the total number of notices (given and not given) is displayed at the end of the analysis.

`topcheck` Integer (1): turns on checking of circuit topology for degeneracies. Valid values are: 0: disable any topology check; 1: a concise report about topology errors and degenerations; 2: a more verbose reporting.

`ignoreshorts` Boolean (0): shorted devices are ignored, that is, they do not contribute to the analyses. On the other hand if set false they are considered in the analysis. This is usefull for example when there are mosfets with drain and source terminals connected together or when there are multi-port controlled sources with one shorted input port. In this case the entire controlled source will be ignored if the "ignoreshorts" option is true.

`unusedmod` Boolean (false): print the name of the models that are defined in the netlist but not referred to by any instance.

`ignoreundefinstpars` Boolean (0): undefined parameters of instances are ignored. This may be usefull to allow the reading of netlists of foreign simulators.

`ignoreundefmodpars` Boolean (0): undefined parameters of models are ignored. This may be usefull to allow the reading of netlists of foreign simulators.

`ignoreundefsubcktparams` Boolean (false): parameters of subckt instances that are not defined in the corresponding subckt body are ignored. This may be usefull to allow the reading of netlists of
oreign simulators.

`ignoreundefparams` Boolean (false): parameters that are not defined (assigned) before their usage are ignored or more precisely return a 0 value.

`ignoreunknownoptions` Boolean (false): unkown options are ignored.

`bounderr` Integer: (-): the severity of the error given when <l> and/or <w> parameters of mosfets are out of bounds. In general these bounds are defined by the <lmin>/<lmax> and <wmin>/<wmax> parameters. Possible choises of this parameter are: 0: none (error is not given) 1: warning (simulation continues) 2: error (simulation is aborted)
Note that if this parameter is not specified, bounds are checked according to the corresponding action defined by the simular parameter of each mosfet model.

`renamedupinst` Boolean (false): rename duplicated instances, that is those instances with names colliding with previously defined ones.

`cc` String: ("cc -O3 -fPIC -shared
`bhvcmp` Boolean: (true): the algebrain expressions of behavioral elements are compiled in a shared library. This shared library is loaded and executed and executed by the simulator.
`temperature` Real (27.0): circuit working temperature in Celsius degrees.
`temp` Real (27.0): circuit working temperature in Celsius degrees.
`subckthier` Boolean (false): print subcircuit hierarchy.
`outintnodes` Boolean (false): when set voltages of internal nodes are saved into the output rawfile. Internal nodes are added by devices, such as mosfets, diodes, bjts and others to deal with 'internal' elements or equations. For example a diode is characterised, at its extenal nodes (those specified in the netlist), by a series connection of a parasitic resistor and the diode itself. The node connecting the parasitic resistor and the diode is considered as internal to the diode model and in general of no meaning to the user (hidden to the user).
`writeparams` Boolean (false): write parameter values.
`digits` Integer (4): print precision.
`width` Integer (-): screen width.
`ascii` Boolean (false): generate ascii rawfile.
`resourceusage` Integer (-): print brief summary of resource usage
`timer` Integer (1800): time interval after which the partial results of the currently running analysis are saved in a temporary file. In case of failure or crash of the analysis, the simulator can be restarted from this saved result.

4.7.2 Alter an instance parameter or simulation option

This statement changes the value of a modifiable device instance parameter for any analyses that follow. Only one parameter can be changed with each alter statement. This statement can be employed also to change a modifiable global simulation option, like temperture. It effects any following analysis. In general, when a device parameter or a global simulation options is changed, the results computed by previous analyses (those that preceded the current alter stetement) become invalid, that is, they can no be futher used to start the following analyses.
ANALYSIS DEFINITION:
Name **alter** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:
`instance` String (-): the name of the instance whose parameter is modified.
`model` String (-): the name of the model whose parameter is modified.
`opt` Boolean (-): specifies that a parameter of simulation options is modified.
`param` String (-): the name of parameter to be altered.
`value` Real (0.0): the new value assigned to the parameter.
`annotate` Integer (3): degree of annotation
`invalidate` Boolean (true): the results computed by the analyses carried out before the application of the current <alter> statement are invalidated. Therefore they can not be further used as a starting guess for the following analyses.

4.7.3 Sweep Analysis

This analysis sweeps parameters of a model, an instance or a simulator option and executes the actions specified in the action body. Sweeps can be applied also to parameters defined through the 'parameter' keyword in the circuit netlist. More that one sweep can be defined in the same sweep card. For example consider

Sweep sweep start=0 stop=1 step=0.1 instance=r1 mioparam="r" begin Tran tran tstop=1m

Shoot shooting autonomous=1 fund=1k restart=no end

A sweep named <Sweep> acts on the parameter <r> of the instance <r1>. Parameter value is linearly swept in the [0,1] interval by steps of value 0.1. At each point of the sweep the <Tran> transient analysis is executed till the <stop=1m> time stop. After the <Tran> analysis the <Shoot> shooting analysis is executed starting from the results computed by the previous <Tran> analysis. In the example above the sweep acts on an instance parameter. Consider the example

parameters p1=0 p2=0 Sweep sweep start=0 stop=1 step=0.1 param="p1" + start=0 stop=1 step=0.1 param="p2" begin Tran tran tstop=1m Shoot shooting autonomous=1 fund=1k restart=no end

the "p1" and "p2" parameters (defined through the 'parameters' keyword) are varied and the same analyses described above are executed for each value assumed by each parameter. Note that this sweep is 'two-dimensional', i.e. the sweep generates a square grid of samples. There is no upper limit to the number of sweeps in the same sweep card.

ANALYSIS DEFINITION:

Name **sweep** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (default is global annotate level):

`resetsweep` Boolean (true): set back all sweeping parameters at the values they had at the beginning of the sweeping analysis. If set to false parameters are left at values assumed at the end of the sweeping analysis.

`center` Real (-): center of sweep

`dec` Integer (-): points per decade

`lin` Integer (50.0): number of points, linear sweep

`log` Integer (50.0): number of points, log sweep

`span` Real (0.0): sweep limit span

`start` Real (0.0): start sweep limit

`step` Real (-): step size, linear sweep

`stop` Real (-): stop sweep limit

`instance` String (-): instance to which parameter sweeping is applied.

`model` String (-): model to which parameter sweeping is spplied.

`opt` Boolean (false): tells that sweeping applyies to a simulator option (specified through the <option> card).

`mioparam` String (-): name of the parameter of the model, instance or simulator option that has to be swept.

`variable` String (-): name of the variable to be swept that has been specified in the body of a <control> analysis or in the body of the <sweep> analysis itself.

`param` String (-): name of the parameter (specified through the keyword 'parameter' in the circuit netlist) to be swept.

4.7.4 Control block

The control analysis interprets a language similar to that of the octave environment. It allows to perform elaborations of the simulation results to alter instance and model parameters and to perform analyses. In other words it gives the possibility to implement algorithms that "drive" the simulation flow. The interpreted language allows the definition of procedures in the same control body. Supported data types are scalars, vectors matrices and array of matrices. Supported types are integer, real and complex. Available functions:

abort: abort the control analysis. abs,

x = arg(z): computes the argument of 'z', defined as 'x = atan2(imag(z)/real(z))' in radiants.

x = conj(y): computes the conjugate of 'y'. If 'y' is real thus 'x = y'.

x = cos(y): computes the cosine of 'y'. The 'y' argument can be a scalar

a vector or a matrix. x = cputime(): get cpu time from the start of the simulator.

diag, eig : computes eigenvalues and eigenvectors of a square matrix; eye,

[d,e] = derivative("func", x, "method", delta): computes the derivative of the "func" scalar function with respect to the 'x' variable. Different "method's can be employed; available ones are "central", which is the default one, "backward" and "forward". The 'delta' argument defines the initial relative step employed to compute the derivative. The estimated value of the derivative is returned as 'd', the absolute error in the estimation is returned in 'e'.

y = db(x): implemetd the 'y=20*log10(x)' function.

fclose, x = elaptime(): get elapsed time from the start of the simulator.

x = fft(y): computes the fast Fourier transform of 'y'. The 'y' argument can be a vector or a matrix. If 'y' is a matrix the Fourier transform is performed column by column and the 'x' result is a matrix. The computed spectrum is monolateral. If the number of samples is even the component at the Nyquist frequency is stored as the last entry of 'x'.

flush() : flush output

fflush(x) : flush the file identified by the 'x' file descriptor.

fd = fopen("filename", "mode"): open a file for read/write accorting to the 'mode; selector. Possible modes are those of standard unix. The file descriptor is returned in the 'fd' variable.

fprintf, x = get("result", index): get simulation results, instance and model parameter values. The "result" string identifies the result in the memory waveform data-base. The syntax is "NameOfTheAnalysis.Eqn" where "NameOfTheAnalysis" refers to a simulator analysis and "Eqn" is the name of a circuit node or branch equation. If a parameter of an instance or model has to be retrived, the syntax is similar to the previous one. The name of the analysis has to be substituted with the name of the instance or model; the node name has to be substituted with the parameter name. If a parameter of an instance does not exist it is considered as a parameter of the corresponding model (associated to the instance) and the search is repeated. The second parameter of get() is an integer; if not supplied the entire result vector is retrieved. If 'index' is equal to 0, the current size of the resulting vector is retrieved. If it is a positive integer the first 'index' elements of the result vector are retrieved. If 'index' is a negative number the last 'index' elements of the result vector are retrieved.

x = gmres("func", y, b, k): the gmres method solves the 'A*x=b' linear equation system. The 'func' function performs the 'A*x' matrix by vector product. The 'y' vector is the initial guess. The 'k' parameter is optional and specifies after how many iterations the gmres method is restarted.

x = ifft(y): computes the inverse fast Fourier transform of 'y'. The 'y' argument can be a vector or a matrix, in this case the Fourier transform is performed column by column and the 'x' result is a matrix.

imag, z = integral(y,x): computes the integral of vector 'y' versus vector 'x'. The lengths of these vectors must be the same. The 'z' returned vector has the same dimension of 'x' and stores the value assumed by the integral versus 'x'.

y = interp1(x0, y0, x, method): interpolates values, where 'x0' stores the known 'x' values, 'y0' stores the known 'y' values, 'x' stores the desired value(s). The length of 'y0' must be equal to that of 'x0'. The 'method' argument is optional, allowed values are "linear" or "bspline".

inv : inverse of a matrix;

y = isnan(x): returns logical variable/array/matrix which is true where the elements of 'x' are 'NaN' values and false where they are not. For example 'isnan([13, Inf, NaN]) => [0, 0, 1, 1]'.

length, [l,u,p] = lu(a) : decomposes 'a' square matrix in the 'l' lower triangular matrix, the 'u' upper triangular one and the 'p' permutation one, such that we have 'l*u=pa'; x = max(a): if

'a' is a vector, 'x' is the (scalar) maximum entry of 'a'. If 'a' is a matrix, 'x' is a vector whose elements are the maximum entries of the corresponding column of 'a'. For example the second entry of 'x' stores the maximum entry in the second column of 'a'.

x = max(a,b): 'a' and 'b' can be scalars/vectors/matrices. If 'a' is a scalar and 'b' a vector, 'a' is compared with each entry of 'b' and the maximum (scalar) value is returned in 'x'. The same happens if 'b' is a matrix. The role of 'a' and 'b' can be exchanged. If 'a' and 'b' are both vectors, 'x' is a vector containing the maximum between the corresponding entries of 'a' and 'b'. If 'a' and 'b' are matrices 'x' is a matrix containing the maximum between the corresponding entries of 'a' and 'b'.

mean, x = min(a): if 'a' is a vector, 'x' is the (scalar) minimum entry of 'a'. If 'a' is a matrix, 'x' is a vector whose elements are the minimum entries of the corresponding column of 'a'. For example the second entry of 'x' stores the minimum entry in the second column of 'a'.

x = min(a,b): 'a' and 'b' can be scalars/vectors/matrices. If 'a' is a scalar and 'b' a vector, 'a' is compared with each entry of 'b' and the minimum (scalar) value is returned in 'x'. The same happens if 'b' is a matrix. The role of 'a' and 'b' can be exchanged. If 'a' and 'b' are both vectors, 'x' is a vector containing the minimum between the the corresponding entries of 'a' and 'b'. If 'a' and 'b' are matrices 'x' is a matrix containing the minimum between the corresponding entries of 'a' and 'b'. For complex number the norm is considered.

x = mktime(tm): convert a time structure corresponding to the local time to the number of seconds since the epoch.

[x,err,iters] = multirootf("func", x, "deriv", iters, abstol): find a solution of a system of nonliear equations by adopting a hybrid version of the Newton iterative method. The 'r=func(x)' function takes as argument the 'x' vector representing the initial value of the solution and returns the corresponding value of the 'r' residue. The 'j=deriv(x)' function takes as input the current value of the solution and returns the Jacobian matrix. If 'deriv' is not specified (argument is omitted) since 'multirootf' is called with only two arguments or the third argument is equal to 'null', the Jacobian matrix is automatically estimated by the Newton hybrid method. The 'iter' variable is optional and defines the maximum allowed number of iterations. The 'abstol' variable is optional and defines the absolute tolerance, when the norm of the residue is below this value, it is assumed that convergence has been reached. The returned values are the 'x' vector that stores the found solution, the real value 'err' that stores the norm of the residue and the integer 'iters' that stores the total number of performed iterations of the Newton algorithm.

norm, x = ones(n,m) : create a matrix of 'n' rows and 'm' column with all entries equal to 1. If the second argument is omitted, a column vector of length 'n' is created.

x = pink(n,slope,fsample) : generate colored noise with spectral density equal to $f^s lope$. The 'n' parameter tells how nany sample are needed. The 'fsample' parameter defines the frequency at which noise samples of colored noise are generated and thus the bandwidth of the generated noise.

x = pinv(y,abs): computes the pseudo-inverse of the 'y' matrix. The 'abs' argument defines the absolute tolerance below which the singular value value of 'y' are not considered. If 'abs' is not given the singular considered are those greater than 's1 * gMachineResolution * max(M,N)' where s1 is the largest singular value, M is the number of rows of 'y' and N is the number of columns

[f,x] = mtpsd(x,bw,nfft,fs) : compute multitaper spectrum estimates of time sequence stored in the 'x' vector; 'bw' is time-bandwidth product its default value is 'bw=1.5'; 'nfft' defines the frequency grid; 'fs' is the sampling frequency.

printf,

[q,r,p] = qr(a): computes the 'q' and 'r' decomposition of the 'a' matrix; entries on the diagonal of 'r' ordered in decreasing magnitude; 'p' is a permutation matrix such that 'q*r-

a*p=0';

x = rand(rows,cols): fills 'x' with pseudo-random elements uniformly distributed between 0 and 1. x = randn(rows,cols): fills 'x' with normally distributed pseudo-random elements having zero mean and variance one. real, x = round(y): return the integer nearest to 'y'. If 'y' is complex, return 'x = round(real(y)) + i*round(imag(y))'.

shift, x = siman("func", x, param): the simulated annealing optimisation method tries to find the minimum of the "func" function. This function has as argument the 'x' vector of parameters versus which the minimum is looked for. The 'func' function must return a positive scale representing the value assumed by the cost functions. The 'param' argument is a vector whose entries are the 7 parameters governing the behaviour of the simulated annealing optimisation method. The measing of each entry follows: param(1): (default 200) How many points do we try before stepping; param(2): (default 1000) How many iterations for each T; param(3): (default 1.0) maximun step size in random walk; param(4): (default 1.0) Boltzmann constant; param(5): (default 0.008) Initial temperature; param(6): (default 1.003) Damping factor for temperature; param(7): (default 2.0e-6) Minimum temperature at which the method stops.

x = simplex("func", x, iters): the simplex optimisation method tries to find the minimum of the "func" function. This function must return a non negative real scalar. The 'x' vector stores the initial guess, 'iters' is the maximum allowed number of iterations that can be performed

x = sin(y): computes the sine of 'y'. The 'y' argument can be a scalar a vector or a matrix.

x = sort(y): sort in ascending order the content of the 'y' real or complex vector.

sprintf, sqrt, std, [x,y] = strptime(str,fmt): convert the string 'str' to the time structure 'x' under the control of the format string 'fmt'. If 'fmt' fails to match, 'y' is 0, otherwise 'y' is set to the position of last matched character plus 1. Always check for this unless you're absolutely sure the date string will be parsed correctly. Supported formats are those from the same C library procedure of Linux;

x = sum(y): adds all elements of the 'y' vector or matrix columnwise; result is the 'x' scalar or row vector, respectively.

svd : computes the singular values and left/right singular vectors of a matrix; system x = transpose(y) : computes the transpose of y. Note this is NOT the conj transpose; the conj transpose operator is represented by the standard "' symbol.

x = zeros(n,m) : create a matrix of 'n' rows and 'm' column with all entries equal to 0. If the second argument is omitted, a column vector of length 'n' is created.

Special variables: anaerror: stores the error code after the execution of an analysis. This variable must be defined and initialised before its usage. If the user does not define this variable or if the error is a fatal one, the error is handled by the simulator. Warnings and notices are discharded. When an error occurs the <anaerror> variable is set equal to 1. A simple usage example follows:

anaerror = 0; Alter alter instance="R1" param="r" value=0; if(anaerror > 0) printf("Alter error: end

The <Alter> analysis chances the value of the <R1> resistor to 0. Since resistance can not be set to 0, an error is generated.

null: this variable is used as argument of functions to specify that an optional argument possibly followed by other ones is not used

ANALYSIS DEFINITION:

Name **control** <*parameter=value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (default is global annotate level): degree of annotation. Values from 1 to 7 turn on display of generic information about the run.

4.7.5 Monte Carlo Analysis

The MonteCarlo analysis is introduce by means of an example. A circuit divider composed of the ‘r1’ and ‘r2’ resistors is considered.

```
ground electrical gnd

parameter rx=10 std=0.5 rmc=rx dev/gaussian=std

v1   n10    gnd   vsource  vdc=10
r1   n10      x   resistor r=rmc
r2     x    gnd   resistor r=rmc

"MonteCarlo analysis" control begin
    count = 0;
    samples(100) = 0;

    Mc mc samples=100 begin
Dc dc print=no annotate=0

count = count + 1;
samples(count) = get( "Dc.x" );
    end

    printf( "\n" );
    printf( "-------------------------------------------------\n" );
    printf( "mean: v(x) = %e std = %e\n", mean(samples), std(samples) );
    printf( "-------------------------------------------------\n" );
    printf( "\n" );

endcontrol
```

The voltage at the ‘x’ output node is half that of the independent voltage source ‘v1’. Value of resistors are equal to the ‘rmc’ parameter. In turn the ‘rmc’ parameter is equal to the ‘rx=10’ one. Any time a parameter is followed by the <lot/process> or <dev/process> keywords a statistical process is assign to the parameter. Therefore the above example ‘rmc’ is linked to a <dev/gaussian> process with variance equal to the value of the ‘std=0.5’ parameter. The ‘control’ analysis named ‘MonteCarlo’ has a body that implements an interpreted language similar to that of Matlab/Octave environments. In particular the ‘count = 0’ statement initialises the ‘count’ variable and the ‘samples(100) = 0’ initialises to 0 the ‘samples’ array. The ‘mc’ MonteCarlo analysis is executed in the control body, that is the random number generators associated to parameters are exercised; at each iteration of the MonteCarlo analysis the related body is executed. In the body a ‘dc’ analysis is first executed, the ‘count’ variable is increased by 1 and then the voltage al node ‘x’ is retrieved and stored in an entry os the ‘samples’ array. When the MonteCarlo analysis is completed the mean value and the standard deviation of the sampled stored in the array are computed and displayed.
ANALYSIS DEFINITION:
Name **mc** *<parameter=value> ...*

ANALYSIS PARAMETERS:

`annotate` Integer (3): degree of annotation.

`clip` Boolean (1): allow clipping of instance and model parameters when their values generated by the ramdom processe go above the maximum allowed value and below the minimum allowed one.

`samples` Integer (100): number of samples considered in the MonteCarlo analysis.

`seed` Integer (-): the seed of the random number generators. If not specified the seed is taken from the system clock.

`maxsigma` Integer (4): define the maximum variance below which a randomly generated number must fall. If the random number is above this variance it is dropped.

4.8 Advanced time domain analyses

4.8.1 Time domain analysis with envelope algorithm

The envelope analysis computes the time domain behavior of both autonomous and non-autonomous circuits. Waveforms made available to the control analysis are:

1) Node voltages and branch currents computed by the envelope analysis along each time window, time samples are stores in the 'time' vector. These waveforms can be accessed through the command 'get("EnvelopeName.node")'.

2) The 'indices' vector that stores the indices of the first sample in each time window. The length of this vector equals the number of time windows of the envelope analysis. Indices can be accessed through 'get("EnvelopeName.indices")'.

ANALYSIS DEFINITION:

Name **envelope** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Degree of annotation. It is an integer number from 1 to 7 that turns on the display of generic information about the analysis execution. Values above 7 are bitwise and turn on specific printouts. In particular:
0x08: print sensitivity matrix and RHS at various phases of the analysis.
0x10: write to an output file the sensitivity matrix at each successful iteration of the envelope algorithm.

`checkstrange` It is a boolean parameter (default false). When it is set true, at each time point of the transient analysis, it is checked if there are some 'strange behaviours' of devices. For example it is checked if bulk-drain and bulk-source parasitic diodes of mosfets are forward biased. This check slows down a bit simulation and can generate a large number of warnings.

`printnodes` Print external/internal node map.

`autonomous` It is a boolean flag (default false) that tells if the circuit is autonomous if set to true. This causes the automatic estimation of the working period of the circuit.

`statevars` It is a list of names of device instances (for example capacitors and inductors) that specify state variables. These state variables are monitored by the procedure that determines the working period of the autonomous circuit. If this list is not given all state variables are monitored

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial condition from which the envelope following analysis is started.

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This

solution is used as initial condition to the envelope following analysis.

`acntrl` Integer (1): select the accuracy control method:

1 relative energy balance; this method computes the electrical energy variation of each capacitor and inductor along each integration interval and the corresponding electrical work delivered by the resistive one-port elements to which they are connected. The electrical work and energy are compared and the integration time step is chosen in order to keep their difference below the user defined threshold;

2 absolute energy balance. This method is identical to the previous one, but the difference between the electrical work and energy is compared to the total energy stored into the corresponding capacitor/inductor. The integration time step is chosen to keep variation with respect to total stored energy below the user defined threshold. In general this method gives less accurate results with respect to the previous one, but the analysis runs faster.

`etoltype` Integer (2): this option specifies the tolerance check adopted to check if the envelope analysis has reached convergence inside a simulation window (one working period of the circuit). Checks are done on the variation of variables computed by the time domain analysis along one time window. Define as 'dX(n)' the variation of the 'n' electrical variable between the beginning and the end of one time window, as 'reltol' and 'abstol' the relative and absolute maximum allowed variations, respectively. The n-th variable has converged if '| dX(n) | < reltol * Y(n) + abstol'. This parameter defines three different ways to compute 'Y(n)', numbered as 1, 2 and 3. Their meanings are listed in the following:

1 : global. 'Y(n)' is defined as the maximum over one all time windows, till the current one, of the maximum value assumed by the 'X(n)' electrical variables, where 'n' varies from 1 to N, being N the number of circuit variables. Note that in this case 'Y(n)' is unique value adopted for all the 'dX(n)' variable variations.

2 : local. 'Y(n)' is defined as the maximum absolute value of 'X(n)' in the time window. Note that in this case 'Y(n)' is different for each electrical variable.

3 : point local. 'Y(n) = | X(n) |'.

The convergence check tightens as the value of this parameter is increased.

`ntoltype` Integer (3): this option specifies the tolerance check adopted at each time point of the analysis to control if the Newton method has reached convergence. Checks are done both on the variation of variables computed by the Newton method and on the entries of the right hand side vector. Define as 'dX(n)' the variation of the 'n' electrical variable, as 'reltol' and 'abstol' the relative and absolute maximum allowed variations, respectively. The n-th variable has converged if '| dX(n) | < reltol * Y(n) + abstol'. The 'ntoltype' parameter defines three different ways to compute 'Y(n)', number as 1, 2 and 3. Their meanings are listed in the following:

1 : global. 'Y(n)' is defined as the maximum over the time interval, till the current time point, of the maximum value assumed by the 'X(n)' electrical variables, where 'n' varies from 1 to N, being N the number of circuit variables. Note that in this case 'Y(n)' is unique value adopted for all the 'dX(n)' variable variations.

2 : local. 'Y(n)' is defined as the maximum absolute value of 'X(n)' in the time interval till the current time point. Note that in this case 'Y(n)' is different for each electrical variable.

3 : point local. 'Y(n) = | X(n) |'.

The convergence check tightens as 'ntoltype' is increased. Default value of this parameter is 2 (local).

`method` Select the integration method: 1 trapezoidal (default); 2 Gear method of various order.

`order` Set the order of the Gear integration method, default value is 2. It has no meaning if other integration methods are employed.

`hgfactor` Set the growing factor of the envelope step H. At each iteration if the local truncation

error is below the threshold set through the <ereltol> and <eabstol> parameters the envelope step is increased by a certain ammount. The maximum allowed increase is set by the estimated local truncation error but the growing factor can be set by the user (default is 2).

`maxh` Maximum allowed value of the envelope integration step H; as default H is unlimited

`corrector` This parameter defines the type of corrector; its value must lie from 0.5 (pure trapezoidal method) to 1.0 (pure Implicit Euler method, default)

`timesteps` List (-): the list containing the integer multiples of the 'period' of the envelope waveform at which a time domain analysis must be performed. In case of autonomous circuits, the 'period' varies and is automatically determined by the envelope analysis. In case of non-autonomous circuits the value of 'period' is given by the user and does not vary during the envelope analysis. In both cases, the envelope integration time step 'H' is adjusted to eaxcly meet the integer multiples specified in the list.

`timeinstants` List (-): the list of time instants at which a time domain analysis must be performed. In case of non-autonomous circuits the value of 'period' is given by the user and does not vary during the envelope analysis. In case of autonomous circuits, the 'period' varies and is automatically determined by the envelope analysis. In both cases, the time domain analyses are performed along time windows whose initial time instants are as close as possible to time instants specified in the list. This is achieved by suitably choosing the envelope integration time step 'H'.

`eabstol` Real (1.0e-6): absolute tolerance to check acceptability of the solution found by the envelope analysis at each working period of the circuit.

`ereltol` Real (1.0e-1): relative tolerance to check acceptability of the solution found by the envelope analysis at each working period of the circuit.

`iabstol` Absolute current tolerance employed by the Newton algorithm at each time point of the time domain analysis (default 1.0e-12).

`ireltol` Relative current tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`vabstol` Absolute voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`vreltol` Relative voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`trabstol` Absolute convergence tolerance of the transient analysis employed by the envelope analysis.

`trreltol` Relative convergence tolerance of the transient analysis employed by the envelope analysis.

`fft` Boolean flag, that if set true, turns on FFT analysis of electrical variables (default false). Fundamental frequency corresponds to the circuit working period, given by the user if the circuit is non-autonomous and computed by the envelope algorithm if the circuit is autonomous.

`fftharms` Number of harmonics considered by FFT

`fftsamples` The number of sampling points in the FFT analysis. FFT analysis ensures that the given number of points is employed, but more points may be employed. This parameters is effective only if it speficies a number of samples grether than '4 * <ffthars> + 1', which represents the minimum allowed number of employed samples.

`floquet` It is a boolen parameter (default false). If set true, the Floquet multipliers related to the jacobian of the state transition matrix computed at convergenge of the shooting method are evaluated.

`lyapunov` It is a boolen parameter (default false). If set true, the Lyapunov exponents are

evaluated.

`maxiter` Integer (100): maximum number of allowed iterations for the envelope algorithm.

`minper` Set the minimum value assumed by the estimated circuit working period. This parameter has meaning only if the <autonomous> parameter is set to true

`cmin` A parasitic capacitor of value <cmin> is added between each node (user given nodes) of the circuit and ground. This is a convergence aid for algorithms. If it set to 0 its effects are turned off

`period` Real (-): the known working periods of the circuit in case of a nonautonomous circuit or an estimation of the working period in cae of autonomous circuits. This estimation is refined by the envelope analysis till reaching the exact working period of the circuit.

`fund` Real (-): the fundamental frequency of the circuit in case of nonautonomous circuits or an estimation of the working frequency in case of autonomous circuit. This value is refined till finding the exact value by the envelope analysis.

`restart` If set to false results obtained by a previous envelope or transient analysis are uses as initial guess to the shooting method

`thermalnet` Use the thermal network

`tinc` Lengthening factor of the time step at convergence. Default value is 1.1.

`tmax` Maximum allowed time step lenght.

`tmin` Minimum allowed time step lenght.

`savelist` It defines a list on nodes or branch currents that will be saved into the output file during simulation. Other computed electrical variables are not saved

`stop` Time stop or ending time instant of the envelope analysis. This parameter is ignored in shooting analysis.

`tstop` Time stop or ending time instant of the envelope analysis. This parameter is ignored in shooting analysis.

`uic` Integer (0): use given initial conditions of inductors and capacitors. Initial conditions are considered in two different ways according to the value of this paramenter; in particular:
1: it is assumed that all the capacitors and inductors have suitable initial conditions. This means that if they are not specified by the user, they are assumed equal to zero. Note that this initial condition determination does not deals with loops of voltages source and capacitors even though consistent initial conditions are given, since the loop current is undertermined. The same applies to inductor cut-sets.
2: it is assumed that all the capacitors and inductors whose initial conditions have not been specified are open circuit and short circuit, respectively. In this case suitable voltage source and capacitor loops orcurrent source and inductor cut-sets do not cause any problem.

4.8.2 Steady state computation through shooting method

Shooting analysis:

it is a time domain analysis that efficiently determines the steady state behaviour of both autonomous and nonautonomous circuits. When dealing with autonomous circuits, the working period is automatically computed by starting from an initial guess given by the user at the beginning of the analysis. When a limit cycle of an autonomous circuit has been found, it is possible to compute the Floquet multipliers that give insight to the local stability of the limit cycle. When the annotation level is suitably set at each iteration of the algorithm some usefull information is printed. For example consider the following line:

|| R* || = 12.992 D[10] R[10] F* = 5.00000000 MHz <4> Max dX = 1.312 m at <N12>
Max RelT = 12.21 m at <N21>

where || R || means the residue, that is, the norm of the right hand side vector of the steady state problem that is solved by employing the Newton iterative method. A possible '*' means

that the sensitivity matrix has been computed, ‘D[10]’ tells that 10 entries of the updating vector computed by the Newton algorithm do not satisfy the convergence criterion, R[10] tells that 10 equations do not meet the convergence criterion, F* = 5.00000000 MHz shows the currently computed working frequency of the circuit, obviously it has meaning only if the circuit is autonomous (oscillator). A possible ‘*’ tells that the algorithm was unable to suitably determine the working period and an old estimation is thus used, eventually <4> shows that the transient analysis has been continued up to 4 times the previously estimated period in order to estimate a new and better one. When the shooting method damps the updating the ‘S = 1.00e-01’ notice appears showing the damping factor, equal to 0.1 in this example. ‘Max dX = 1.312 m’ is the maximum voltage difference obtained by subtracting voltages at the beginning and at the end of the circuit working period. An indication of the circuit node at which the maximum difference has been computed is also given. ‘Max RelT = 12.21 m’ is the maximum relative error is computing the initial conditions that leads to the steady state working condition of the circuit. The circuit node/branch that yields tha maximum relative error is displayed. The algorithm computes also the eigenfunctions related to the variational model of the circuit at steady state. The variational model is characterised by the fundamental matric M that admits the ‘v(T)’ left eigenvectors and the ‘u(T)’ right eigenvectors. The ‘v(t)’ and ‘u(t)’ eigenfunctions along one working period ‘T’ of the circuit are determined starting from the value of eigenvectors at the beginning of the period. With respect to the state formulation these eigenfunctions are determined for each equation of the MNA formulation (referring to nodes and currents of current controlled elements). If the circuit is autonomous the eigenvector and thus the derived eigenfunction associated to the Floquet multiplier equal to 1 are automatically identified. In general there can be more than one Floquet multiplier very close to 1 (think of high Q oscillators); that found as the representative of the Floquet multiplier theoretically equal to 1 is marked by the ‘*’ symbol in the multiplier printout. The number of the Floquet multipliers differs from the number of state variables. This happens since the modified nodal analysis formulation is adopted in the simulator. Floquet multipliers refer to node voltages of voltage controlled elemets and branch currents of current controlled elements (capacitors and inductors for example). If a capacitor is connected between nodes ’k’ and ’j’ two Floquet multipliers are computed one for node voltage ’k’ and the other for node voltage ’j’. The ’vB(t)’ eigenfunctions refer to the product of the ‘v(t)’ ones with the ‘B(t)’ function. The ‘B(t)’ function represent the sensitivity of pseudo-state variables (for example node voltages) with respect to current injected into the node (the same comment can be aplied to branch currents). For more theoretical details see

M. Farkas, ‘Periodic Motions, Springer-Verlag‘, 1994.

A. Demir, A. Mehrotra and J. Roychowdhury, ‘Phase Noise in Oscillators: A Unified Theory and Numerical Methods for Characterization‘, IEEE Trans. on CAS-I, Vol. 47, No. 5, May 2000, pp. 655-674.

A. Brambilla, P. Maffezzoni, G. Storti-Gajani, ‘Computation of Period Sensitivity Functions for the Simulation of Phase Noise in Oscillators’, IEEE Trans. on CAS-I, Vol. 52, No. 4, April 2005.

Waveforms that can be stored in memory and made available to the control analysis are:

1) Node voltage and branch current waveforms can be retrieved through the statement ’get("ShootName.node")’, time is stored in the ’time’ vector that can be retrieved through ’get("ShootName.time")’

2) Node voltage and branch current spectra can be retrieved through the statement ’get("ShootName.node:spectrum" The frequency samples are stored in the ’freq’ vector. It can be retrived through the statement ’get("ShootName.freq").

3) device current waveforms, the related time samples can be retrieved in the ’devtime’ vector.

4) Floquet periodic eigenfunctions that can be retrived through the statement 'get("ShootName.u1@n1"), where 'u1' is the type of eigenfunction and 'n1' is the referring circuit node, the related time samples are stored in the 'eigtime' vector.

ANALYSIS DEFINITION:
Name **shooting** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Degree of annotation. It is an integer number from 1 to 7 that turns on displaying of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts. In particular:
0x08: print sensitivity matrix and RHS at various phases of the shooting analysis.
0x10: at convergence of the shooting method the fundamental matrix is saved in a file named as the analysis.
0x20: give a short report on actions performed to find saltation matrices.

`maxiters` Integer (100): maximum allowed iterations for the shooting algorithm.

`restart` Boolean (yes): restart the shooting analysis, that is, do not use previous results computed by a transient analysis, an envelope analysis or a shooting analysis as initial guess to the current shooting analysis. If the circuit is autonomous the period computed by one of the previous analyses is adopted as initial guess. If the circuit is non-autonomous and the <"period"> parameter has not been specified the period specified or computed by the previous analyses is adopted.

`printnodes` Print external/internal node map.

`printsh` Boolean (no): when true the solution obtained at convergence of the shooting method is printed.

`savelist` It defines a list on nodes or branch currents that will be saved into the output file during simulation. Other computed electrical variables are not saved

`thermalnet` Boolean (false): use the thermal network.

`checkstrange` Boolean (1): when true at each time point of the transient analysis, it is checked if there are some 'strange behaviours' of devices. For example it is checked if bulk-drain and bulk-source parasitic diodes of mosfets are forward biased. This check slows down a bit simulation and can generate a large number of warnings.

`autonomous` Boolean (no): it tells if the circuit is autonomous. In this case the working period of the circuit is automatically estimated.

`vprobe` List (-): the list containing the pair of nodes at which the voltage probe is connected. If the second node of the list is omitted, it is considered connected to ground. The format is 'vprobe=["pos","neg"]'.

`probeharm` Real (1): the index of the harmonic at which the probe drives the circuit.

`vprobemax` Real (-): the maximum value of the magnitude of the voltage probe employed during the sweep to find the steady state solution.

`eabstol` Real (1.0e-6): absolute tolerance employed by the shooting analysis to check convergence.

`ereltol` Real (1.0e-4): relative tolerance employed by the shooting analysis to check convergence.

`iabstol` Real (1.0e-12): absolute current tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`ireltol` Real (1.0e-6): relative current tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`vabstol` Real (1.0e-6): absolute voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`vreltol` Relative voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`trabstol` Real (1.0e-12): absolute convergence tolerance at each time point of the transient analyses performed during the execution of the shooting analysis.

`trreltol` Real (1.0e-3): relative convergence tolerance at each time point of the transient analyses performed during the execution of the shooting analysis.

`ntoltype` Integer (2): option specifies the tolerance check adopted at each time point of the analysis to control if the Newton method has reached convergence. Checks are done both on the variation of variables computed by the Newton method and on the entries of the right hand side vector. Define as 'dX(n)' the variation of the 'n' electrical variable, as 'reltol' and 'abstol' the relative and absolute maximum allowed variations, respectively. The n-th variable has converged if '| dX(n) | < reltol * Y(n) + abstol'. The 'ntoltype' parameter defines three different ways to compute 'Y(n)'. Their meanings are listed in the following:

1 : global. 'Y(n)' is defined as the maximum over the time interval, till the current time point, of the maximum value assumed by the 'X(n)' electrical variables, where 'n' varies from 1 to N, being N the number of circuit variables. Note that in this case 'Y(n)' is a unique value adopted for all the 'dX(n)' variable variations.

2 : local. 'Y(n)' is defined as the maximum absolute value of 'X(n)' in the time interval till the current time point. Note that in this case 'Y(n)' is different for each electrical variable.

3 : point local. 'Y(n) = | X(n) |'.

The convergence check tightens as 'ntoltype' increases.

`acntrl` Integer (1): select the accuracy control method:

1 relative energy balance; this method computes the electrical energy variation of each capacitor and inductor along each integration interval and the corresponding electrical work delivered by the resistive one-port elements to which they are connected. The electrical work and energy are compared and the integration time step is chosen in order to keep their difference below the user defined threshold;

2 absolute energy balance. This method is identical to the previous one, but the difference between the electrical work and energy is compared to the total energy stored into the corresponding capacitor/inductor. The integration time step is chosen to keep variation with respect to total stored energy below the user defined threshold. In general this method gives less accurate results with respect to the previous one, but the analysis runs faster.

`skip` Integer (1): skip time point (advance in time) in case of unrecoverable convergence failures of the Newton algorithm. It helps to continue and successfully terminate the simulation even though accuracy can be locally compromised. Possible choises of this parameter are:

0: no skipping is applyied in case there is no convergence of the Newton algorithms;

1: skipping is applyed but only if there is convergence of the Newton algorithm in the new time point, determined after the forward time skipping.

2: forward time skipping is applyed anyway, even though there is no convergence of the Newton algorithm in the new time point.

`cmin` Boolean (false): if true, parasitic capacitors of suitable values are inserted between each circuit node and ground when the simulator is in trouble to compute the solution. This can aid integration algorithms.

`maxdeltav` Real (-): the maximum allowed variation of node voltage allowed during the iterations of the Newton algorithm. This should aid convergence and avoid possible overflow.

`jump` Boolean (false): activate finding of discontinuities in the vector field. Saltation matrices are exploited to construct a correct fudamental matrix of circuits characterised by

discontinuities in the vector field. Discontinuities in the vector field can be also introduced by sharp varying functions modeling circuit elements, such as for example the 'y=tanh(a*x)' function with the 'a' gain too large. The solver automatically determines possible discontinuities due to numerical problems introduced by too sharp nonlinear functions and automatically introduce 'saltation' matrices. The solver introduces also 'saltation' matrices due to dscontinuities caused by commutations of pseudo analog-to-digital and digital-to-analog converters during simulations of mixed digital/analog circuits and due to behavioural elements tagged as digital.

`jumpabstol` Real (1.0e-6) absolute tolerance employed to determine possible jumps in the vector field (current through capacitors and voltage across inductors).

`jumpreltol` Real (1.0e-3) relative tolerance employed to determine possible jumps in the vector field (current through capacitors and voltage across inductors).

`floquet` Integer (0): the Floquet multipliers related to the state transition matrix computed at convergence of the shooting method are evaluated. Possible values of this parameter are: 1 : the Floquet multipliers are shown after convergence of the shooting algorithm; 2 : the Floquet multipliers are shown at each iteration of the shooting algorithm.

`printeig` Boolean (no): print the left and right eigenvectors of the monodromy matrix.

`eigf` Boolean (false): the periodic eigenfunctions of the Floquet analysis are computed. In particular the uj(t) and vj(t) eigenfunctions related to the pseudo-state variables of the MNA formulation are computed. Furthermore the vjB(t) functions that relate possible small signal perturbations applied to the circuit nodes to the pseudo-state variables and the Muj(t) eigenfunctions that relate the pseudo-state variables to the node potentials and device branch currents (of those elements that are not voltage controlled) are also computed.

`samples` Integer (200): the number of samples used to compute the sensitivity functions (Floquet eigenfunctions) with respect to pseudo-state variables along one working period of the circuit. Floquet eigenfuncitons are computed in a reduced number of time points with respect to those used by the variable time step length integration algorithms employed by the time domain analysis. This is done mainly to reduce the total ammount of memory needed to store and elaborate the results. The minimum allowed number of samples is equal to 100.

`printmo` Boolean (no): when true the fundamental matrix is printer. In case of autonomous circuits the fundamental matrix is bordered with an extra column and an extra row that refer to the of the solution with respect to the working period of the circuit.

`fmnumber` Integer (32766): The maximum number of the Floquet multipliers that are considered for computing the u(t) and v(t) eigenfunctions. Floquet multipliers are ordered from the largest towards the smallest, the first <fmnumber> are considered.

`devcurr` Boolean (no): the currents of devices, for example the drain current of mosfets, the anode-cathode current of diodes, are stored in the output file of the Floquet analysis.

`eigfnorm` Boolean (false): normalise the Floquet eigenfunctions. Each eigenfunction is rescaled by the corresponding Floquet exponent; this results in a periodic function. Note that largely dumped or increasing eigenfunction must be multiplied by relatively large or small scaling factors mainly at the end of the period and this can lead to amplification of truncation errors. The result is that the Floquet eigenfunction is no longer period since truncatin errors sum up in an umpredictable way to the correct Floquet eigenfunction.

`fft` Boolean (no): it turns on the computation of Fourier series coefficients of the electrical variables. The value of the fundamental frequency is derived from the working period of the circuit that is that given by the user in the non-autnomous case or that computed by the shooting algorithm in the autonomous case.

`fftharms` The number of harmonics considered during the computation of the Fourier series coefficients. This number coincides with that of the computed coefficients.

`fftsamples` The number of sampling points adopted to compute the coefficients of the Fourier series. It is ensured that at least the specified number of points are used but more points can be employed. The coefficients are computed by evaluating the Fourier integrals and not by applying the FFT algorithm. This ensures better results even though few coefficients are computed. This parameters is effective only if it specifies a number of samples greater than '4 * <ffthars> + 1', which represents the minimum allowed number of employed samples.

`method` Integer (1): select the integration method:
1 trapezoidal; 2 Gear methods of order from 1 to 6.

`order` Integer (2) set the order of the Gear integration method. It has no meaning if other integration methods are employed.

`minper` Set the minimum value assumed by the estimated circuit working period. This parameter has meaning only if the <autonomous> parameter is set to true

`uic` Integer (0): use given initial conditions of inductors and capacitors. Initial conditions are considered in two different ways according to the value of this parameter; in particular:
1: it is assumed that all the capacitors and inductors have suitable initial conditions. This means that if they are not specified by the user, they are assumed equal to zero. Note that this initial condition determination does not deals with loops of voltages source and capacitors even though consistent initial conditions are given, since the loop current is under-determined. The same applies to inductor cut-sets.
2: it is assumed that all the capacitors and inductors whose initial conditions have not been specified are open circuit and short circuit, respectively. In this case suitable voltage source and capacitor loops orcurrent source and inductor cut-sets do not cause any problem.

`period` Real (0.0): if the circuit is autonomous it is an estimation of the working period; if the circuit is non-autonomous it represents the fixed working period of the circuit and it is not computed by the shooting analysis. If the initial guess is loaded from a file storing results results by a previous shooting analysis and this parameter is omitted, the period is set equal to that of the previous loaded analysis. If the initial guess is taken from a previous shooting analysis, successfully terminated before the current one and this parameter is omitted, the period is set equal to that of the previous analysis.

`fund` Real (-): the working frequency of the circuit in case of nonautonomous circuits or an estimation of the working frequency in case of autonomous circuits. This estimation is refined till computing the exact working frequency by the shooting analysis.

`freltol` Real (1.0e-4): relative tolerance employed by the shooting analysis to check convergence in determining the working frequency of autonomous circuits.

`tinc` Real (1.1): lengthening factor of the time step at convergence.

`tmax` Maximum allowed time step length.

`tmin` Minimum allowed time step length.

`saman` Boolean (no): turns on/off the samanskii method. At each iteration of the shooting method the sensitivity matrix of electrical variables at the end of the shooting interval with respect to the initial conditions at the beginning of the interval is computed. Even though it is a by product of the transient analysis, at each time point a matrix by matrix product is needed and it costs CPU time mainly when a large number of electrical variables are involved. At each iteration of the shooting method the norm of the residue is monitored; when it monotonically decreases of a relative factor lower than 0.9 the computation of the sensitivity matrix is suspended; at each subsequent iteration of the shooting method the same previous sensitivity matrix is adopted. This leads to an approximate Newton iteration,

but it saves CPU time even though, in general, a larger number of transient analyses are needed.

`damping` Real (1.0): the initial value of the damping factor employed by the Newton algorithm to update the new initial conditions. Normally at the beginning the Newton iterations are undamped, for particular circuit it can be useful to start with damped iterations in order to aid and speed up convergence.

`statevars` It is a list of names of device instances (for example capacitors and inductors) that specify state variables. These state variables are monitored by the procedure that determines the working period of the autonomous circuit. If this list is not given all state variables are monitored

`solver` Integer (0): the employed solver; available ones are:

0 : direct solver based on the Newton method and the LU factorisation of the fundamental matrix;

1 : direct solver based on the Newton method and the QR factorisation of the fundamental matrix.

2 : direct solver based on hybrid-Newton method.

3 : iterative solver based on the Newton method and the 'gmres' method.

As simple guidelines to choose a solver one has to consider that in general the direct solvers are faster on small and medium size circuits, the iterative solver largely outperformes the direct one on large circuits. Circuits with more than a thousand nodes are considered as large; in this case the iterative solver is automatically chosen if this option is not specified, that is, if a solver is not directly specified by the user through this option. The iterative solver requires tens time less memory than direct solvers. If direct solvers are chosen they take a smaller amount of memory if the Gear integration method is employed. The QR factorisation method of the fundamental matrix is preferred to the LU one, since there are stiff circuits where some pseudo-state variables yield Floquet multipliers extremely close to 1. These Floquet multipliers indicate that the fundamental matrix is singular. Singularities can be handled by QR factorisation, since the columns of the Q matrix can span the subspace into which the solution 'lives'. If in these cases the LU factorisation method is chosen there are convergence difficulties of the shooting method since the residue can not be suitably reduced.

2 : iterative solver based on hybrid Newton and 'saltation' matrices. It is suited to the computation of the steady state solution of problems where electrical variables shows discontinuities of the first kind. Note that mixed analog/digital circuits fall in this class.

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial condition from which the shooting algorithm is started.

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This solution is used as initial guess to the shooting algorithm.

4.8.3 Steady state computation through the multi-tone shooting method

Experimenal analysis; use at your own risk! What it does will be detailed in a future release.

ANALYSIS DEFINITION:

Name **mtshooting** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Degree of annotation. It is an integer number from 1 to 7 that turns on displaying of generic information about the analysis run. Values above 7 are bitwise and turn on specific

printouts. In particular: 0x08: print sensitivity matrix and RHS at various phases of the envelope analysis or of the shooting method.

`acntrl` Select the accuracy control method: 1 energy balance. This method computes the electrical energy variation of each capacitor and inductor along each integration interval and the corresponding electrical work delivered by the resistive one-port elements to which they are connected. The electrical work and energy are compared and the integration time step is chosen so that to keep their difference below the user defined threshold (this is the default accuracy mechanism); 2 energy balance. This method is identical to the previous one, but the difference between the electrical work and energy is compared to the total energy stored into the corresponding capacitor/inductor. The integration time step is chosen to keep variation with respect to total stored energy below the user defined threshold. In general this method given less accurate results with respect to the previous one, but the analysis runs faster.

`cmin` A parasitic capacitor of value <cmin> is added between each node (user given nodes) of the circuit and ground. This is a convergence aid for algorithms. If it set to 0 its effects are turned off

`delay` Real (-): the time interval along which the time domain analyses (transient analyses) are performed. These transient analyses are carried out by starting from the given initial conditions and compute the solutions at the end of each simulation interval. The solutions at the end of each time interval are also computed by adopting the dalay operator in the frequency domain. It relates each solution at the end of a time interval to those at the beginning of all the time intervals.

`eabstol` Absolute tolerance employed by the multi-tone shooting analysis (default 1.0e-12).

`ereltol` Relative tolerance employed by the multi-tone shooting analysis (default 1.0e-3).

`iabstol` Absolute current tolerance employed by the Newton algorithm at each time point of the time domain analysis (default 1.0e-12).

`ireltol` Relative current tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`maxiter` Maximum number of allowed iterations for the multi-shooting algorithm (default 100).

`maxdeltai` Real (-): maximum allowed variation of current of state variables at each iteration of the Newton algorithm. If variation of any of the state variabled exceeds the maximum allowed value, the Newton step is suitably shrinked, preserving direction.

`maxdeltav` Real (-): maximum allowed variation of voltage of state variables at each iteration of the Newton algorithm. If variation of any of the state variabled exceeds the maximum allowed value, the Newton step is suitably shrinked, preserving direction.

`method` Select the integration method: 1 trapezoidal (default); 2 Gear method of various order.

`order` Integer (2): set the order of the Gear integration method. It has no meaning if other integration methods are employed.

`harms` Integer (2): largest considered harmonic of the first and second frequency.

`printnodes` Print external/internal node map.

`restart` If set to false results obtained by a previous envelope or transient analysis are uses as initial guess to the shooting method

`spectra` Compute spectra of time domain waveforms.

`time` Boolean (0): output time domain waveforms by starting from the computed spectra.

`points` Integer (1000): number of time points in output waveforms.

`thermalnet` Use the thermal network

`tinc` Real (1.1): lengthening factor of the time step at convergence.

`tmax` Maximum allowed time step lenght.

`tmin` Minimum allowed time step lenght.

`trabstol` Absolute convergence tolerance of the transient analysis employed by the envelope analysis.

`trreltol` Relative convergence tolerance of the transient analysis employed by the envelope analysis.

`savelist` It defines a list on nodes or branch currents that will be saved into the output file during simulation. Other computed electrical variables are not saved

`uic` Integer (0): use given initial conditions of inductors and capacitors. Initial conditions are considered in two different ways according to the value of this paramenter; in particular:
1: it is assumed that all the capacitors and inductors have suitable initial conditions. This means that if they are not specified by the user, they are assumed equal to zero. Note that this initial condition determination does not deals with loops of voltages source and capacitors even though consistent initial conditions are given, since the loop current is undertermined. The same applies to inductor cut-sets.
2: it is assumed that all the capacitors and inductors whose initial conditions have not been specified are open circuit and short circuit, respectively. In this case suitable voltage source and capacitor loops orcurrent source and inductor cut-sets do not cause any problem.

`vabstol` Absolute voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`vreltol` Relative voltage tolerance employed by the Newton algorithm at each time point of the time domain analysis.

`saman` Boolean (false): turns on/off the samanskii method.

4.8.4 Periodic Small Signal Analysis

The pac analysis computes transfer functions of circuits that admit a periodic solution. These circuits can be both autonomous (oscillators) and non-autonomous. The pac analysis linearises the circuit along the limit cycle defined by the periodic large signal solution. Non-linear elements are transformed in time-varying linear ones that thus exibit frequency translations. The pac analysis must be preceeded by a shooting analysis that determined the limit cycle of the unperturbed circuit. Then the effects of small signal perturbations (assumed additive) are computed by the pac analysis.
ANALYSIS DEFINITION:
Name **pac** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (-): degree of annotation. Values from 1 to 7 turn on display of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08 : printout the state transition matrix.

`start` Start sweep limit

`stop` Stop sweep limit

`center` Center of sweep

`span` Sweep limit span

`step` Step size, linear sweep

`lin` Number of points, linear sweep

`dec` Points per decade

`log` Number of points, log sweep

`sweeptype` Integer (0): type of frequency sweep. Possible ones are:
0: absolute sweep; 1: relative sweep. 2: absolute offset sweep. 3: relative offset sweep. When 'absolute sweep' is chosen the frequency is swept in the interval defined exclusively

by the sweep. When the 'relative sweep' is chosen the frequency is swept in the interval specified by the sweep with respect to the fundamental frequency corresponding to the working period determined by the large signal steady state analysis. In other words the frequency of the sweep adds to the circuit large signal fundamental. When the 'absolute offset sweep' is chosen the frequency is swept as described in the 'absolute sweep' mode but the results are referred to the frequency of the 'offset sweep', i.e. the output frequency does not consider the fundamental and harmonics of the circuit. When the 'relative offset sweep' is chosen the frequency is swept as described in the 'relative sweep' mode but the results are referred to the frequency of the 'relative sweep'.

`refh` Integer (1): the harmonic to which a relative sweep refers to.

`freq` Real (-): the frequency value in single frequency analysis.

`print` Boolean (false): print the result in single frequency analysis.

`harms` Real (10): the number of harmonics of the large signal spectra of the unperturbed circuit that are considered in the analysis.

`samples` Integer (-): the number of time points employed to sample the large signal steady state solution and to define the size of the related variational model. When not specified, it is derived from the number of harmonics defined by the "harms" parameter as '4 * "harms"'.

`decomp` Boolean (false): when true the resulting component of the spectra along the 'u1(t)' eigenfunction and those obtained by difference with it are computed. These components have meaning only for autonomous circuits. The component along the 'u1(t)' eigenfunction is defined as 'phase modulation' and that obtained by subtracting this component from the complete result is defined as 'amplitude modulation'.

`tmax` Real (-): maximum integration time step allowed during the transient analysis related to the <pac> analysis. Before performing a <pac> analysis the circuit is linearised along one working period. The solution along this working period is determined by a transient analysis. The <tmax> parameter upper limits the maximum integration time step that can be used by the transient analysis.

4.8.5 Noise analysis of a periodic circuit

The pnoise analysis computes the noise at the specified output (for example a node pair) of circuits that admit a periodic solution. These circuits can be both autonomous (oscillators) and non-autonomous. The pnoise analysis linearises the circuit along the limit cycle defined by the periodic large signal solution. Non-linear elements are transformed in time-varying linear ones that thus exibit frequency translations. The pnoise analysis must be preceeded by a shooting analysis that determines the limit cycle of the circuit. The folding of noise sources is considered and thus the total noise at the specified output is composed of various contributions due to up-conversion and down-converions of noise. The number of beats considered in folding noise can be specified through an analysis parameter. An example of a pnoise analysis card is:

Pnoise pnoise start=50k stop=4M lin=100 onodes=["out"]

The 'Pnoise' analysis determines the total noise at the "out" node of the circuit, a linear sweep of the frequency, composed of 100 samples, is started at a frequency offset from the fundamental equal to '50k'. The sweep ends at a frequency far '4M' from the fundamental. The value of the fundamental is automatically determined by the 'shooting' analysis preceeding the 'pnoise' one. A 'pnoise' analysis can be carried out also at specific frequency offsets given through a list. In this case, if the spectrum of the solution computed by the previous 'shooting' analysis has been determined, the noise computed by the 'pnoise' analysis is displayed both in 'dB' and in 'dBc'. The pnoise analysis can also decompose the computed total noise in the phase and amplitude components. The phase noise is defined as the projection of the total noise along the eigenfunction corresponding to the Floquet multiplier theoretically equal to 1. The amplitude

noise is defined as the difference between the total noise and the phase noise. Waveforms that can be stored in memory and made available to the control analysis are the computed noise spectrum and its phase and amplitude components. These vectors can be retrieved through the statements 'get("AnalysisName.out")', 'get("AnalysisName.out:pm")' and 'get("AnalysisName.out:am")'. The frequency axis can be retrived through the stateemnt "get("AnalysisName.freq")'.

ANALYSIS DEFINITION:
Name **pnoise** <*parameter=value*> ...

ANALYSIS PARAMETERS:

`annotate` Degree of annotation. It is an integer number values from 1 to 7 turn on displaying of generic informations about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08: printout the state transition matrix. 0x10: save the state transition matrix in a file named as the analysis.

`start` Start sweep limit

`stop` Stop sweep limit

`center` Center of sweep

`span` Sweep limit span

`step` Step size, linear sweep

`lin` Number of points, linear sweep

`dec` Points per decade

`log` Number of points, log sweep

`sweeptype` Integer (2): type of frequency sweep. Possible ones are:
0: absolute sweep; 1: relative sweep. 2: offset sweep
When 'absolute sweep' is chosen the frequency is swept in the interval defined exclusively by the sweep. When the 'relative sweep' is chosen the frequency is swept in the interval specified by the sweep with respect to the fundamental frequency corresponding to the working period determined by the large signal steady state analysis. In other words the frequency of the sweep add to the oscillator fundamental. When the 'offset sweep' is chosen the frequency is swept as in the 'relative sweep' but the results are referred to the frequency of the sweep. This parameter acts also in single frequency analysis define by the <"freq"> parameter.

`refh` Integer (1): the harmonic to which a relative sweep refers to.

`freq` Real (-): the frequency value in single frequency analysis. This parameter must not be specified contemporary to the frequency sweeping command.

`onodes` List (-): the list containing the pair of nodes (the ground node can be omitted) at which the noise is computed. The format is <onodes=["node1","node2"]>.

`harms` Real (7): the number of harmonics of the large signal spectra of the unperturbed circuit that are considered in this analysis.

`samples` Integer (-): the number of time points employed to sample the large signal steady state solution and to define the size of the related variational model. When not specified, it is derived from the number of harmonics defined by the <"harms"> parameter.

`type` Integer (7): types of noise sources that are considered during the analysis. Possible choices are:
0x1 : thermal noise; 0x2 : shot noise; 0x4 : flicker noise.
The value of this parameter can be the result of the bitwise-or of types of noise sources.

`decomp` Boolean (false): when true the resulting component of the noise spectra along the 'u1(t)' eigenfunction and those obtained by difference with it are computed. These components have meaning only for autonomous circuits. The component along the 'u1(t)'

eigenfunction is defined as 'phase noise' and that obtained by subtracting this component from the complete result is defined as 'amplitude noise'.

`relnf` Real ("ALU resolution" dB): this parameter defines the relative noise flore that can be introduced by the pnoise analysis. Components of the transfer functions below this value are not considered.

`solver` Integer (-): type of solver employed in the pnoise analysis; available ones are:
0: direct solver 1: iterative solver.
If the solver is not specified the same one employed in the related shooting analysis is adopted.

`tmax` Real (-): maximum integration time step allowed during the transient analysis related to the <pnoise> analysis. Before performing a <pnoise> analysis the circuit is linearised along one working period. The solution along this working period is determined by a transient analysis. The <tmax> parameter upper limits the maximum integration time step that can be used by the transient analysis.

4.9 Advanced frequency domain analyses

4.9.1 Pole Zero Analysis

This analysis linearises the circuit about the DC operating point and determines the poles and zeros of the linear circuit. In particular to compute zeros, a transfer function must be defined by specifying the input source and the output voltage or current.
ANALYSIS DEFINITION:
Name **pz** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (default is global annotate level): degree of annotation. Values from 1 to 7 turn on display of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08 : print MNA matrix; 0x10 : dump MNA matrix in the "analysis_name.m" file.

`statematrix` Boolean (false): dump (pseudo)state matrix (only with direct solver).

`thermalnet` Boolean (false): use the thermal network.

`hzout` Boolean (true): Output pole/zero values as Hz or as equivalent eigenvalues

`solver` Integer (2): type of solver (solvers) to be used by the analysis. Possible values are:
1 : iterative solver; 2 : direct method; 3 : iterative solver followed by the direct one.
In general the iterative solver is faster than the direct one, but it can be less accurate. The direct one is not suited for large circuits, since it can take too much cpu time. Note that the iterative solver is unable to compute all the poles of a large circuit. The maximum number of poles in N-2 where N is the number of poles characterising the circuit. The computed poles are ordered from that with the largest real part towards that with the smaller one.

`maxm` Real (1.0e12): poles or zeros are not considered when their module is greater than this parameter. Recall that modified nodal analysis formulation introduces poles and zeros at infinity for circuit nodes of the algebraic portion of the DAE.

`maxp` Integer (-): the maximum number of poles with largest real part that are computed by the iterative solver. This parameter has meaning only when the iterative solver is employed.

4.9.2 Periodic Steady-State Analysis

The analysis computes the periodic steady state response of autonomous and nonautonomous circuits in the frequency domain by applying the Harmonic Balance Method. MemWaveforms made available to control blocks:

1) vector of employed fundamental and harmonics can be accessed by through the 'freq' variable.

2) spectrum of each node voltage and branch equation can be eccessed by postnoming the node/branch equation name to the analysis name.

ANALYSIS DEFINITION:
Name **harmonic** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:

`annotate` Integer (3): degree of annotation. Values from 1 to 7 turn on displaying of generic informations about the analysis run. Values above 7 are bitwise and turn on specific printouts and in particular:
0x08: print some information about the convergence of the Newton method;
0x10: print MNA matrix and RHS vector at each iteration of the Newton method.
0x20: at each iteration of the Newton method the MNA matrix is written in a file named as the analysis.
0x40: write the Floquet matrix in a file named as the analysis.

`time` Output time domain signals

`maxiters` Integer (100): allowed maximum number of iterations.

`restart` Boolean (true): restart the analysis, that is, do not use previous solution as initial guess.

`printnodes` Print internal nodes numbers

`solver` Integer (0): the solver employed to compute the solution. Available solvers are:
0: direct solver (block LU factorisation)
1: iterative solver (gmres).

`ta` Boolean (false): this parameter turns on and off the time assisted harmonic balance solver.

`method` Select the integration method used in the time assisted harmonic balance:
1 trapezoidal (default); 2 Gear method of various order.

`order` Set the order of the Gear integration method used in the time assisted harmonic balance, default value is 2. It has no meaning if other integration methods are employed

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This solution is used as initial guess to the analysis.

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial condition from which the analysis is started.

`fund` Real (-): the fundamental frequency at which the circuit works. If the considered circuit is autonomous, this value gives an initial estimate of the final working frequency computed by the harmonic balance analysis.

`period` Real (-): the period at which the circuit works If the considered circuit is autonomous, this value gives an initial estimate of the final working period computed by the harmonic balance analysis.

`harms` Integer (2): the number of harmonics employed by the harmonic analysis.

`thresh` Real (1.0e-15): during the harmonic analysis each nonlinear or time varying element is characterised by a spectrum that contributes off-diagonal entries to the block-organised MNA matrix. The numerical effort to LU factorise this block-matrix at each iteration of the Newton algorithm depends on the bandwidth of each block sub-matrix. In general the magnitude of entries lower as they are far from the main diagonal (high order harmonics). This parameter defines a relative threshold with respect to the largest magnitude of the spectrum components of each device, below which the entries can be neglected. A suitable

choice of <thresh> can improve the speed of the harmonic balance method. On the other hand, it introduces also inaccuracy and even, it can prevent convergence, that is, it causes a drop in the effectivenes and efficiency of the harminic balance method. Therefore it must be carefully used.

`saman` Real (2.0): samanskii constant. At each iteration of the Newton algorithm the norm of the residue (right hand side vector) is computed and compared to that computed at the previous iteration. If the ratio between the two norms is less than the value of the samanskii constant plus one then the Jacobian is recomputed. In general Jacobian recomputation speeds up convergence.

`autonomous` Integer (0): solve an autonomous circuit. The following different methods to find a steady state solution can be used:
1: continuation;
2: continuation with no hybrid newton acceleration;
4: hybrid newton.
5: conventional
The 'continuation' solver is bases on the insertion of a probe in the circuit. It is composed of different phases. In the first phase the magnitude of the probe is swept starting from 0 till a suitable maximum value that depeneds on the power supply of the oscillator. During this sweep the current through the voltage probe or the branch voltage across the current probe is monitored. If the real part or the imaginary part of these electrical variables crosses the zero value, the sweep is interrupted and the solution is refined to find the exact amplitude of the probe that cases the branch current of the voltage probe or the branch voltage across the current probe to be 0. This solution is then continued by sweeping the amplitude or the frequency of the voltage probe untill the modulus of the branch current of the voltage probe or the branch voltage across the current probe is 0. When this happens a solution of the steady state problem has been found. Note that the insertion of the voltage probe can be automatic, that is the method automatically determines the nodes to which connect the voltage probe.
The 'continuation with no hybrid newton acceleration' acts as the 'continuation' method by does not apply the hybrid newton to try to accelerate convergence to a steady state solution. In fact hybrid newton does not converge with some circuits and thus wasts cpu time.
The 'newton' solver is based on the insertion of a voltage probe or a current probe in the circuit. It adopts the Newton iterative method to find the 0 value of the branch current of the voltage probe or of the branch voltage of the current probe. The unkowns are the magnitude and the frequency of the probe. In general convergence is achieved if the Newton method is started from a suitable initial condition. It is suggested to perform a pole/zero analysis and to employ the frequency of the unstable pole as initial guess.
The 'conventional' solver adds the working frequency of the autonmous circuit among the problem unknowns and uses the Newton iterative method to solver the enlarge harmonic balance problem. This method can not be employed by itself, since it needs an initial guess. This guess can be obtained for example by a previous harmonic analysis that employs a probe but that does not solve it or by time domain analyses such as trasient, envelope, shooting.

`freltol` Real (1.0e-6): the relative error tolerance in the computation of the working frequency of autonomous circuits.

`fabstol` Real (1.0e-12): the absolute error tolerance in the computation of the working frequency of autonomous circuits.

`floquet` Boolean (false): compute the Floquet multipliers of the circuit.

`flqharms` Real (-): the number of harmonics employed to determine the Floquet multipliers.
`clsminsize` Integer (-): the minimum number of elements that have to populate a cluster in order to have the cluster itself considered as meaningful. During clustering of Floquet exponents, there can be clusters composed of a small number of elements with respect to the total number of harmonics. This is caused by spectrum truncation and round-off errors. This parameter introduces a threshold so that these clusters are ignored.
`vprobe` List (-): the list containing the pair of nodes at which the voltage probe is connected. If the second node of the list is omitted, it is considered connected to ground. The format is <vprobe=["pos","neg"]>.
`vprobemag1, ...` Real (0.0): the magnutude of the voltage probe.
`vprobeph1, ...` Real (0.0): the phase of the voltage probe.
`vprober1, ...` Real (0.0): the value of the resistor connected in series to the voltage probe.
`vprobemax` Real (-): the maximum value of the magnitude of the voltage probe employed during the sweep to find the steady state solution.
`iprobemax` Real (-): the maximum value of the magnitude of the current probe employed during the sweep to find the steady state solution.
`gminstepping` Boolean (flase): turn on and off the gmin stepping continuation method.
`ggroundstepping` Boolean (flase): turn on and off the gground stepping continuation method.
`srcstepping` Boolean (flase): turn on and off source stepping.
`initstep` Init source stepping value
`minstep` Minimum step allowed in source stepping
`ireltol` Relative current convergence criterion
`iabstol` Absolute current convergence criterion
`vreltol` Real (1.0e-6): relative voltage convergence criterion
`vabstol` Real (1.0e-6): absolute voltage convergence criterion

4.9.3 Almost-Periodic Steady State Analysis

The analysis computes the periodic steady state solution of a circuit. Both non-autonomous and autonomous circuits are supported. Results made available to the <control> analysis are: voltage node and branch current spectra (output spectra), employed fundamental and beats (keyword to access them is <freq>), matrix of indices for the retrival of fundamentals and beats inside the <freq> vector (keyword to access it is <freqindices>). The DC component is the first element of index 1 in the vector of output spectra. Specific beat can is defined by the expression

f(i,j) = (i-1)*F1 + (j-1)*F2

where ‘i’ is the index referring to the first fundamental ‘F1’ and ’j’ is the index refering to the second fundamental ‘F2’. For example by assuming that the name of the spectral analysis is <Sp>, the index of the second fundamantal can be retrieved inside the body of a control analysis through the statements

indices = get("Sp.freqindices"); i=1; j=2; idx2 = indices(i,j) + 1;

Note that index ‘i’ and index ’j’ have been increased by 1 and the same has been done for ‘idx2’ since alements of the ‘indices’ matrix are numbered by starting from (1,1).

ANALYSIS DEFINITION:
Name **spectral** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:
`annotate` Integer (5): degree of annotation. Values from 1 to 7 turn on displaying of generic informations about the analysis run. Values above 7 are bitwise and turn on specific printouts. In particular: 0x08: print some information about the convergence of the Newton method; 0x10: print MNA matrix and RHS vector at each iteration of the Newton

method. 0x20: at each iteration of the Newton method the MNA matrix is written in a file named as the analysis. 0x40: print matrices related to probes. This option has meaning only when the probe insertion technique is used.

`printnodes` Print internal nodes numbers

`restart` Boolean (true) or String (-): restart the analysis, that is, do not use previous solution as initial guess. In the other case (boolean false) the result from the previous analysis are used as initial guess. If the name of a previous analysis is given, the results computed by this analysis are used as initial guess, therefore it can be not necessarily the previous one.

`load` String (-): the path of the file from which a previously saved solution is read. The file format can be both ascii and bz2 and it is automatically determined during loading. This solution is used as initial guess to the spectral algorithm.

`save` String (-): the path of the file into which the solution is saved (bz2 format is adopted). The solution stored in this file can be later loaded by the simulator and used as initial condition from which the spectral algorithm is started.

`saman` Real (2.0): samanskii constant. At each iteration of the Newton algorithm the norm of the residue (right hand side vector) is computed and compared to that computed at the previous iteration. If the ratio between the two norms is less than the value of the samanskii constant plus one then the Jacobian is recomputed. In general Jacobian recomputation speeds up convergence.

`srcstepping` Apply source stepping

`minstep` Minimum allowed step in source stepping

`maxiter` Max number of iterations

`gminstepping` Boolean (flase): turn on and off the gmin stepping continuation method.

`ggroundstepping` Boolean (flase): turn on and off the gground stepping continuation method.

`fund1` First fundamental frequency

`fund2` Second fundamental frequency

`harms` Integer (1): largest considered harmonic of the first and second fundamental.

`harms1` Integer (1): largest considered harmonic of the first fundamental.

`harms2` Integer (1): largest considered harmonic of the second fundamental.

`trunc` Integer (0): the truncation schema used in DFT to define the components of the spectrum considered in the analysis. Available choices are:
0: square; 1: diamond.

`time` Boolean (0): output time domain signals.

`interval` Real (-): time domain output interval in seconds.

`points` Integer (200): number of time points in output waveforms.

`vabstol` Real (1.0e-6): Absolute voltage convergence criterion.

`vreltol` Real (1.0e-6): Relative voltage convergence criterion.

`iabstol` Real (1.0e-12): Absolute current convergence criterion

`ireltol` Real (1.0e-6): Relative current convergence criterion

`autonomous` Integer (0): solve an autonomous circuit. The following different methods to find a steady state solution can be used:
1: continuation;
3: continuation with the insertion of only one probe;
4: hybrid newton;
5: conventional.
The 'continuation' solver is bases on the insertion of a probe in the circuit. It is composed of different phases. In the first phase the magnitude of the probe is swept starting from 0 till a suitable maximum value that depeneds on the power supply of the oscillator. During this sweep the current through the voltage probe or the branch voltage across the current

probe is monitored. If the real part or the imaginary part of these electrical variables crosses the zero value, the sweep is interrupted and the solution is refined to find the exact amplitude of the probe that cases the branch current of the voltage probe or the branch voltage across the current probe to be 0. This solution is then continued by sweeping the amplitude or the frequency of the voltage probe untill the modulus of the branch current of the voltage probe or the branch voltage across the current probe is 0. When this happens a solution of the steady state problem has been found. Note that the insertion of the voltage probe can be automatic, that is the method automatically determines the nodes to which connect the voltage probe.

The 'newton' solver is based on the insertion of a voltage probe or a current probe in the circuit. It adopts the Newton iterative method to find the 0 value of the branch current of the voltage probe or of the branch voltage of the current probe. The unkowns are the magnitude and the frequency of the probe. In general convergence is achieved if the Newton method is started from a suitable initial condition. It is suggested to perform a pole/zero analysis and to employ the frequency of the unstable pole as initial guess.

The 'conventional' solver adds the working frequency of the autonmous circuit among the problem unknowns and uses the Newton iterative method to solver the enlarge harmonic balance problem. This method can not be employed by itself, since it needs an initial guess. This guess can be obtained for example by a previous harmonic analysis that employs a probe but that does not solve it or by time domain analyses such as trasient, envelope, shooting.

`freltol` Real (1.0e-6): the relative error tolerance in the computation of the working frequency of autonomous circuits.

`vprobe1` List (-): the list containing the pair of nodes at which the voltage probe on the first fundamental is connected. If the second node of the list is omitted, it is considered connected to ground. The format is <vprobe1=["pos","neg"]>.

`vprobe1mag1, ...` Real (0.0): the magnitude of the voltage probe acting of the first fundamental.

`vprobe1ph1, ...` Real (0.0): the phase of the voltage probe acting on the first fundamental.

`vprobe1r1, ...` Real (0.0): the value of the resistor connected in series to the voltage probe acting on the first fundamental.

`vprobe1max` Real (-): the maximum value of the magnitude of the voltage probe employed during the sweep to find the steady state solution.

`vprobe2` List (-): the list containing the pair of nodes at which the voltage probe on the second fundamental is connected. If the second node of the list is omitted, it is considered connected to ground. The format is <probe1=["pos","neg"]>.

`vprobe2mag1, ...` Real (0.0): the magnitude of the voltage probe acting of the second fundamental.

`vprobe2ph1, ...` Real (0.0): the phase of the voltage probe acting on the second fundamental.

`vprobe2r1, ...` Real (0.0): the value of the resistor connected in series to the voltage probe acting on the second fundamental.

`vprobe2max` Real (-): the maximum value of the magnitude of the voltage probe employed during the sweep to find the steady state solution.

`vprobe2freq1t` Integer (0): the index of the first fundamental or harmonic that beats with the frequencies and harmonics of the second probe related to the second fundamental.

4.9.4 Small Signal S-Parameter Analysis

This analysis linearizes the circuit about the DC operating point and performs a small signal analysis. If desired, the analysis will be performed repeated while sweeping some independent

variable. The variable may be either frequency or some device instance parameter. Not all device instance parameters are modifiable, and some of those that will change the DC operating point when modified. If changing a parameter affects the DC operating point, it will be recomputed on each step. The modified parameter is reset to its original value after the analysis.
ANALYSIS DEFINITION:
Name **sp** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:
`annotate` Integer (default is global annotate level): degree of annotation. Values from 1 to 7 turn on display of generic information about the analysis run. Values above 7 are bitwise and turn on specific printouts:
0x08 : print MNA matrix; 0x10 : dump MNA matrix in the "analysis_name.m" file.
`freq` Real (-): the frequency value when a device value parameter is swept or the frequency value in single frequency analysis. This parameter must not be specified contemporary to the frequency sweeping command.
`port` Active port, may be repeated
`file` S-parameters output file name
`start` Start sweep limit
`stop` Stop sweep limit
`center` Center of sweep
`span` Sweep limit span
`step` Step size, linear sweep
`lin` Number of points, linear sweep
`dec` Points per decade
`log` Number of points, log sweep
`sweep` Device whose parameter value is to be swept
`param` Name of parameter to be swept

4.9.5 Noise analysis of a periodic circuit

Noise analysis computes the total noise at the specified port of a circuit. Noise is computed by linearising the circuit in the operating point, therefore a DC analysis is done before any noise analysis if a valid operating point is not available, yet. The thermal, shot and fliker sources of noise in circuit elements are considered. Note that since the circuit is linearised in the operating point, currents of elements have constant values.
ANALYSIS DEFINITION:
Name **noise** <*parameter*=*value*> ...

ANALYSIS PARAMETERS:
`annotate` Degree of annotation. It is an integer number values from 1 to 7 turn on displaying of generic informations about the analysis run.
`start` Start sweep limit
`stop` Stop sweep limit
`center` Center of sweep
`span` Sweep limit span
`step` Step size, linear sweep
`lin` Number of points, linear sweep
`dec` Points per decade
`log` Number of points, log sweep
`freq` Real (-): the frequency value in single frequency analysis. This parameter must not be

specified contemporary to the frequency sweeping command.

`onodes` List (-): the list containing the pair of nodes (the ground node can be omitted) at which the noise is computed. The format is <onodes=["node1","node2"]>.

Bibliography

Books

[CDK87] L.O. Chua, C. A. Desoer, and E. S. Kuh. *Linear and Nonlinear Circuits*. New York: McGraw-Hill, Editions, 1987 (cited on page 12).

[Far94] Miklós Farkas. *Periodic motions*. New York, NY, USA: Springer-Verlag New York, Inc., 1994. ISBN: 0-387-94204-1 (cited on page 33).

[Kuz95] Y. A. Kuznetsov. *Elements of Applied Bifurcation Theory*. Springer-Verlag, 1995 (cited on page 33).

[Rhe95] R. W. Rhea. *Oscillator Design and Computer Simulation*. New York: McGraw-Hill, 1995 (cited on page 30).

[VS83] J. Vlach and K. Singhal. *Computer Methods for Circuit Analysis and Design*. Van Nostrand Reinhold Company, 1983 (cited on page 12).

Articles

[BBS11] F. Bizzarri, A. Brambilla, and G. Storti Gajani. "Phase Noise Simulation in Analog Mixed Signal Circuits: An Application to Pulse Energy Oscillators". In: *Circuits and Systems II: Express Briefs, IEEE Transactions on* 58.3 (2011), pages 154–158. ISSN: 1549-7747. DOI: 10.1109/TCSII.2011.2111570 (cited on page 35).

[Biz+] Federico Bizzarri et al. "Simulation of real world circuits: extending conventional analysis methods to circuits described by heterogeneous languages". In: *Circuits and Systems Magazine, IEEE (Forthcoming)* () (cited on page 35).

[BGS09] A. Brambilla, G. Gruosso, and G. Storti Gajani. "Determination of Floquet Exponents for Small-Signal Analysis of Nonlinear Periodic Circuits". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 28.3 (2009), pages 447–451. ISSN: 0278-0070. DOI: 10.1109/TCAD.2009.2013285 (cited on page 32).

[BGS10] A. Brambilla, G. Gruosso, and G. Storti Gajani. “FSSA: Fast Steady-State Algorithm for the Analysis of Mixed Analog/Digital Circuits”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 29.4 (Apr. 2010), pages 528–537. DOI: 10.1109/TCAD.2010.2042886 (cited on page 32).

[BMS05] A. Brambilla, P. Maffezzoni, and G. Storti Gajani. “Computation of period sensitivity functions for the simulation of phase noise in oscillators”. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 52 (4 Apr. 2005), pages 681–694 (cited on page 32).

[BS03] A. Brambilla and G. Storti-Gajani. “Frequency Warping in Time Domain Circuit Simulation”. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 50 (7 July 2003), pages 904–913 (cited on pages 26, 27).

[Bra+10] A. Brambilla et al. “Improved Small-Signal Analysis for Circuits Working in Periodic Steady State”. In: *Circuits and Systems I: Regular Papers, IEEE Transactions on* 57.2 (Feb. 2010), pages 427–437. ISSN: 1549-8328. DOI: 10.1109/TCSI.2009.2019403 (cited on page 32).

[HRB75] Chung-Wen Ho, Albert E. Ruehli, and Pierce A Brennan. “The modified nodal approach to network analysis”. In: *Circuits and Systems, IEEE Transactions on* 22.6 (June 1975), pages 504–509. ISSN: 0098-4094. DOI: 10.1109/TCS.1975.1084079 (cited on page 12).

Technical Reports

[MP01] Ron Mancini and Richard Palmer. *Sine-Wave Oscillator*. Technical report. Texas Instruments, Mar. 2001 (cited on page 27).

Index

D

Downloading, Installing 7, 37

E

envelope 25

F

Frequency Warping example 24

R

Restart option (`tran`) 23

S

shooting 27